MAPL Design Guide

If you've ever tried to design a medium to large sized sequential state machine or controller in a high density PAL®-type architecture or programmable gate array, you'll be familiar with "product term limited," "routing," and "skew" problems. By the time you've dealt with these issues you've probably paid a major penalty in overall system performance.

National Semiconductor, already a leading supplier of quality, high speed EECMOS, TTL, and ECL programmable logic devices, is proud to introduce a new family of higher-density EECMOS programmable logic devices, designed specifically to address the needs of high speed controllers and state machines.

Our new Multiple Array Programmable Logic (MAPL™) family of EECMOS devices will perform at system speeds of up to 50 MHz, or more if feedback is not required. The MAPL devices offer a choice of 28-, 44- or 68-pin PLCC packages to suit your design requirements and conserve valuable board space.

In providing flexible, user-programmable architectures that utilize standard PLD development tools and programmer support, we hope that our high performance MAPL family of devices will help you to maintain your competitive edge and win the time-to-market race.

Rick Walker
Strategic Marketing Manager
Programmable Logic

Introduction

This databook introduces you to National Semiconductor's new MAPL family of higher density programable logic devices. The book is not only a compilation of datasheets for the associated products, but also includes a software overview of our new OPAL™ PLD development software, a detailed application section and finally a list of current third-party programming support hardware.

All MAPL devices contain a core architecture of multiple interconnected FPLA pages, that is expandable into more resourceful and larger pin-count packages. More importantly, the delay time from any input to any output via any internal FPLA page is always the same for any given MAPL device. Finally, a single MAPL device is able to integrate multiple PAL, GAL, and discrete logic devices.

The first generation MAPL 1 series datasheets cover all aspects and switching characteristics of the MAPL128 and MAPL144 devices. While the MAPL 2 series advance information sheets give an overview of the second generation MAPL244 and MAPL268 devices.

MAPL Product Family

Device	Pins	Macro Cells	Max Input Pins	Max Output Pins	Speed with f/back (MHz)
MAPL1 Series:					
MAPL128	28	27	21	16	45.5
MAPL144	44	27	21	24	45.5
MAPL2 Series:					
MAPL244	44	64	32	32	50
MAPL268	68	64	36	32	50

Table of Contents

CHAPTER 1

MAPL128 Datasheet

MAPL128, MAPL144
Multiple Array Programmable Logic

PRELIMINARY
March 1991

General Description

The MAPL128 and MAPL144 are the first in a new series of higher density, electrically erasable CMOS programmable logic devices based on a proprietary National Semiconductor programmable logic array architecture. The MAPL128 and MAPL144 products integrate multiple Field Programmable Logic Array (FPLA) device types. The architecture consists of multiple PLAs that are interconnected via programmable macrostate registers and a global input bus. This multiple PLA architecture imitates a large, continuous FPLA, allowing easy implementation of complex state machines, controllers, microinstruction sequencers and general synchronous logic designs. The devices also allow for easy and cost effective integration of multiple TTL counter, register and logic functions.

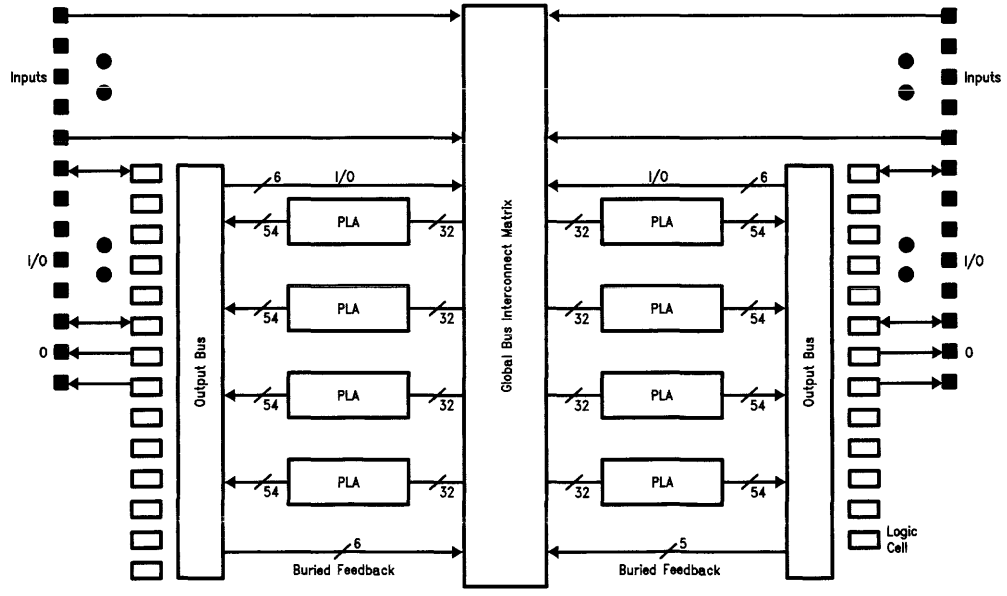
The MAPL™ product family is supported by National Semiconductor's Open Programmable Architecture Language (OPAL™) software package and by other third-party software packages that utilize a common open PLA file format. OPAL provides for total control of logic utilization and partitioning, within the MAPL devices.

Programming of these devices is accomplished using readily available, industry standard, PLD programming equipment. NSC guarantees a minimum 100 erase/write cycles.

The MAPL128 and MAPL144 are available in 28-pin and 44-pin, PLCC packages respectively. These packages conform to the JEDEC standard power, ground and clock pin placements.

Features

- High density programmable logic array architecture
- 45 MHz system performance
- Replaces multiple PAL/PLA devices
- Electrically erasable CMOS technology:
  - — 100% functionally tested
  - — Instantly reconfigurable logic
  - — Minimum of 100 erase/write cycles
- Low power: I_{CC} = 140 mA max
- 27 programmable macrocells
  - — Choice of DE, JK, RS or T type registers
- Asynchronous reset capability
- Fully supported by National's OPAL development software
- Industry standard programmer support
- Security cell prevents copying



TL/L/11146–2

FIGURE 1. MAPL128

Functional Description

The MAPL product family solves performance and product term limitation problems associated with standard PAL and PLA architectures. The product densities allow for integration of multiple PLA device types, thereby reducing power consumption and board space while increasing system reliability. The MAPL family is ideally suited to state machine designs and offers considerable performance advantages, for Mealy and Moore state machine applications, over other programmable solutions. An extensive multi way branch capability allows for input or state variables to transition to a maximum of 65,536 different possible states.

The MAPL128 and MAPL144 incorporate a flexible macrocell architecture that implements both D-type (with clock enable) and JK registers. This alleviates the need for software transformations or additional product terms to implement these functions.

Dynamic Logic Allocation

MAPL devices utilize a proprietary architecture which dynamically allocates user-defined product and sum terms. The allocation of logic is dependant upon the logic design requirements. This active partitioning architecture solves the performance, power consumption and product term limitations associated with traditional PAL, GAL, PLA and partitioned PAL architectures. By determining the state or input dependency of each logic term required, only the necessary logic terms will be active and consume power. In fact, the power consumption of the MAPL128 and MAPL144 products is close to that of a single standard EECMOS GAL device. The reduction in active sense amps reduces power consumption and internal noise without affecting system design applications. Either input or state variables can actively allocate the required terms for the additional logic arrays with zero performance delay. The same product terms required for normal output and state transitions are also used to control the PLA allocation or loading of the next PLA array. An extension of each PLA is used for interconnecting the partitioned PLAs. This array extension drives three registers called global macrostate registers. The logic partitioning and PLA allocation can be accomplished, automatically or manually, by utilizing these user-defined global macrostate feedback registers.

Array Description

The MAPL128 and MAPL144 each comprise eight individual FPLA arrays, interconnected with three global macrostate registers. The macrostate registers are available on the global input bus, which feeds the inputs of all of the FPLA arrays (see *Figures 14* and *15*). Each FPLA, having both a programmable AND array and a programmable OR array, offers a significant amount of flexibility over PAL type devices. Unlike PAL architectures, the product terms in a PLA architecture can be shared among multiple outputs and state bits. Duplication of product terms for each output or state transition is not required, since the OR array is user programmable. The FPLA type architecture in the MAPL product family, offers the maximum product term utilization for additional efficiency and speed, in state machine applications. The MAPL devices allow for easy integration of multiple PAL, GAL, FPLA and PROM based architectures.

TABLE I. MAPL Product Family

Product	MAPL128	MAPL144
Total Product Terms	132	132
Package Pins	28	44
Total Outputs	16	24

MAPL128 Product Description

The MAPL128 is functionally equivalent to a large, continuous FPLA. Having a total of 128 product terms (16 x 8 pages), the MAPL128 is the largest FPLA currently available. Additional programmable sum terms provide state feedback onto a global input bus, which is available to all of the FPLA pages. The device is available in a 28-pin PLCC package, with JEDEC standard power, ground and clock pin placements.

The MAPL128 has eight independent FPLA planes, each having the same 58 x 16 x 54 configuration. Each AND array has a total of 58 true and complimentary inputs. The 16 product terms provide inputs to the programmable OR array. These 16 transition terms can be programmed and interconnected to any of the 54 OR terms. Six of the OR terms are used for the control of the global macrostate registers. The remaining 48 sum terms feed inputs to 24 user configured macrocell registers. The macrostate register outputs have fixed feedback paths to the global input bus, to determine which PLA page is enabled. Each PLA page is a completely programmable entity, similar to any ordinary PLA product. Of the 24 user defined macrocell registers, 8 are buried, 4 are dedicated outputs and 12 are bidirectional I/O.

TABLE II. MAPL128 Pin Description

Product	MAPL128
Power and GND Pins	2
Dedicated Clock	1
Dedicated Enable	0
Dedicated Inputs Pins	9
Maximum Inputs Pins	21
Dedicated Outputs Pins	4
I/O Pins	12
Maximum Output Pins	16
Total Pins	28
Global Macro-State Registers	3
Buried Registers	8
Total Output Registers	16
Total Registers	27
Total AND Array Inputs	32
Total OR Array Outputs	54
Logic Transition Terms	128
Control Terms	4
Sum Terms	432
Total Product and Sum Terms	564

Connection Diagram

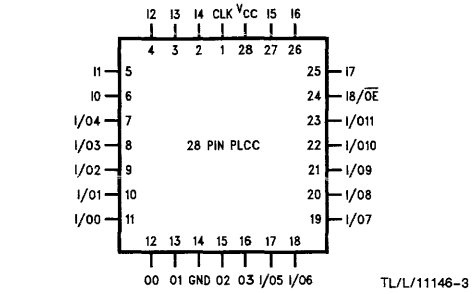


FIGURE 2. MAPL128 Pinout

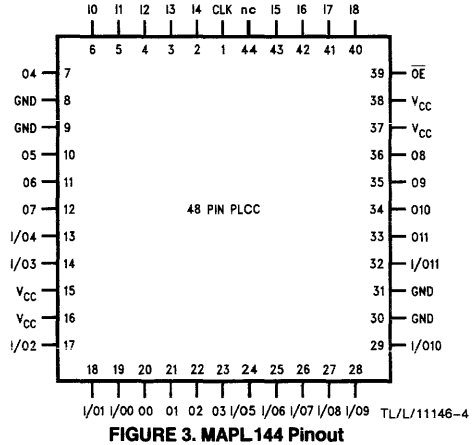


FIGURE 3. MAPL144 Pinout

MAPL144 Product Description

The MAPL144 has the same FPLA structure and size as the MAPL128. However, the MAPL144 is housed in a 44-pin PLCC, allowing an additional 8 dedicated output pins, a dedicated output enable pin, as well as the 44-pin JEDEC standard multiple power and ground pins (4 of each).

TABLE III. MAPL144 Pin Description

Product	MAPL144
Power and GND Pins	8
Dedicated Clock	1
Dedicated Enable	1
Dedicated Inputs Pins	9
Maximum Inputs Pins	21
Dedicated Outputs Pins	12
I/O Pins	12
Maximum Output Pins	24
Total Pins	44
Global Macro-State Registers	3
Buried Registers	8
Total Output Registers	24
Total Registers	27
Total AND Array Inputs	32
Total OR Array Outputs	54
Logic Transition Terms	128
Control Terms	4
Sum Terms	432
Total Product and Sum Terms	564

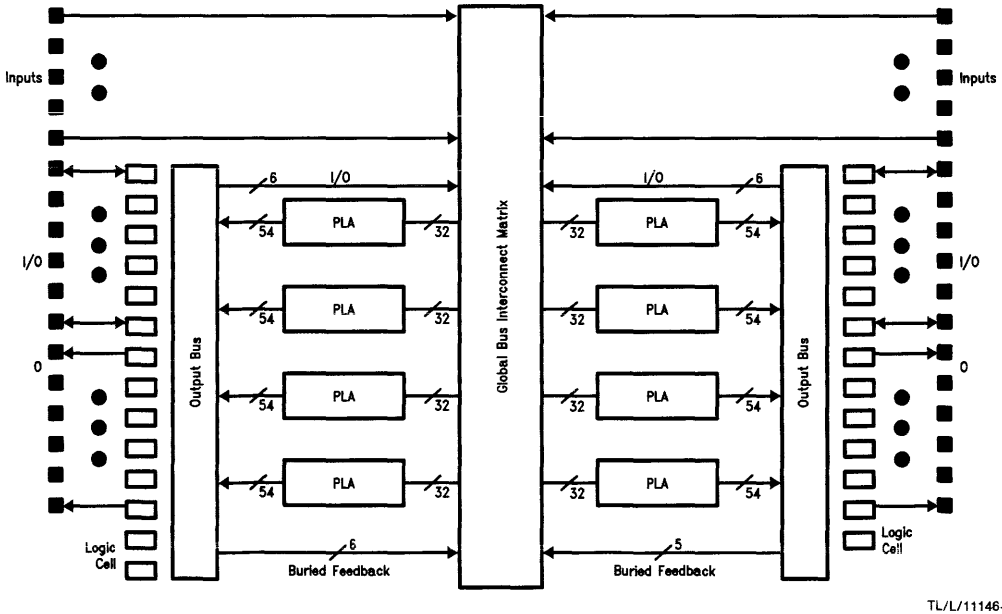


FIGURE 4. MAPL144

Logic Macrocell

The MAPL products include a flexible user configurable macrocell structure. The same registered macrocell is utilized for I/O, buried and output functions. Each macrocell can be configured as a true JK or D type register with clock enable. Both register types are implemented in hardware and do not require software transformations or additional product terms to implement the functions. All buried, global, I/O and output register macrocells are clocked on the rising edge of clock.

JK registers allow for implementation of RS and T registers. JK registers efficiently implement large state machines and counters. Programmable polarity is available on both J and K terms allowing "else" conditions to hold, set, reset or toggle registers which minimize the requirement for additional product terms. The following table assumes non inverting J and K terms.

TABLE IV. JK Type Macrocell

JK Truth Table			
Function	**J**	**K**	**Q(n + 1)**
Hold	0	0	Q(n)
Reset	0	1	0
Set	1	0	1
Toggle	1	1	!Q(n)

RS Truth Table			
Function	**S**	**R**	**Q(n + 1)**
Hold	0	0	Q(n)
Reset	0	1	0
Set	1	0	1
Avoid	1	1	X

T Truth Table		
Function	**T**	**Q(n + 1)**
Hold	0	Q(n)
Toggle	1	!Q(n)

Any one of the macrocells can be programmed to select the alternative D type output register. The D type output allows easy implementation of data registers. This register type also maintains complete compatiblity with most PAL devices. As with the JK register, there is independent polarity control on both the D input term and the clock enable term. When the enable register input, E, is low, the register contents are unchanged regardless of CLK and D inputs.

TABLE V. DE Type Macrocells

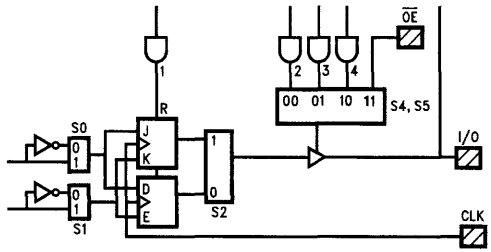
DE Type Macrocell			
Function	**E**	**D**	**Q(n + 1)**
Hold	0	X	Q(n)
New	1	0	0
New	1	1	1

All macrocell types have user programmable global control features. All register truth tables are dependent on the initialization term being inactive. The global term allows for asynchronous reset of all macrocells. This term asynchronously initializes all registers independent of present state, macrostate or inputs. The registers will hold the initialized data until the next valid, enabled clock, changes the data.

All output macrocells have flexible output enable control which allows for selection from up to three global output enable product terms or a dedicated output enable ($\overline{OE}$) pin. The dedicated $\overline{OE}$ pin offers AC performance advantages in TRI-STATE® delays.

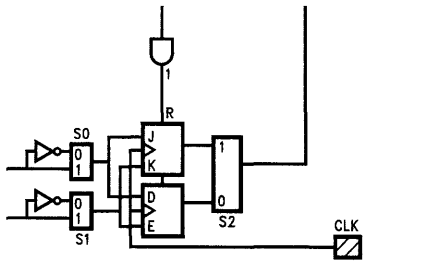
Each output register can be active low or active high. This allows I/O feedback or register output to be programmable active low or active high. This allows designs to have all registers type and both pin polarities, as shown in *Figure 7.*

Input/Output Macrocell



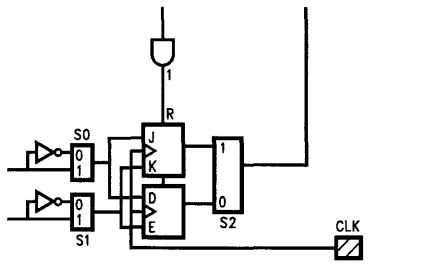
TL/L/11146–6

Buried Macrocell



TL/L/11146–7

Global State Macrocell

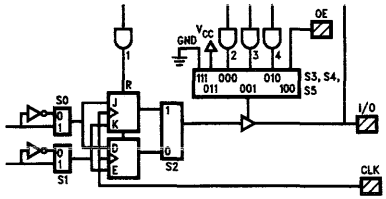


TL/L/11146–8

FIGURE 5. MAPL128 Macrocells

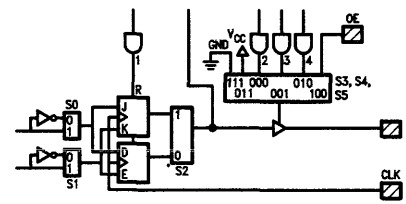
Logic Macrocell (Continued)

Input/Output Macrocell



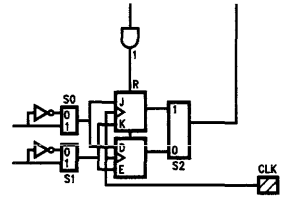
TL/L/11146–9

Output/Buried Macrocell



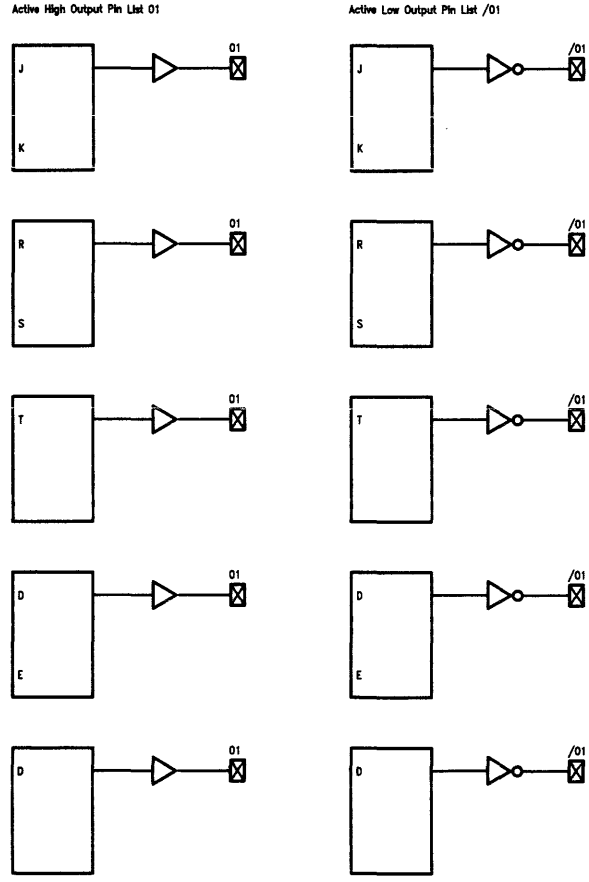
TL/L/11146–10

Global State Macrocell



TL/L/11146–11

FIGURE 6. MAPL144 Macrocells



TL/L/11146–5

FIGURE 7

DC Specifications

Specifications are guaranteed over the recommended operating conditions only.

DC parameter description:

V_{OL}: Voltage specification with 16 outputs held low and loaded at max I_{OL} level specified.

V_{OH}: Voltage specification with 16 outputs held high and loaded at max I_{OH} level specified.

V_{IH}, V_{IL}: Input high voltage min and input low voltage max specifications.

I_{IH}, I_{IL}: Current specification for all input and I/O pins and include forcing voltage conditions.

$I_{OS}(I_O)$: Current specifications sets a single output in a high state, forces zero volts (0V) and reads current. The specification disables the outputs not tested. The test is for a maximum duration of one second.

I_{CC}: Current specification includes V_{CC}, temp, frequency of operation, number of outputs active and output loading. I_{CC} specified with open outputs, F(w/feedback) = 25 MHz.

The worst case power consumption calculation is dependent on external capacitive loads, frequency of operation and number of active inputs and outputs.

AC Specifications

The following data summarizes critical AC specifications. The AC specifications are based on simulation results using worst case commercial V_{CC}, temperature extremes and array loading. AC load will be equivalent to existing PAL and GAL product standards.

AC parameter description:

t_{SU}: Data input or buried feedback to output or buried register input. Register set up time.

t_{CLK}: Output register clock to valid data output.

t_{CYCLE}: Clock period with feedback.

t_H: Data hold time for output register clock.

t_{PZX}, t_{PXZ}: Input to product term output enable, output disable.

t_{DZX}, t_{DXZ}: Dedicated $\overline{OE}$ pin input to output enable, output disable.

t_{IPD}: Propagation delay from pin input or buried register to initialization of register.

t_{IRC}: Recovery time from inactive input to valid rising edge of clock; minimum time from reset recovery input to clock rising edge.

t_{CH}: Clock width high; minimum clock high to clock low.

t_{CL}: Clock with low. Minimum clock low to clock high.

f_{MB}: F_{max} with buried feedback or input. Maximum clock frequency determined; having clock transition at buried register feedback, to any register, or input to any register. $1/(t_{SU})$

f_{MO}: F_{max} with I/O feedback; maximum clock frequency determined; having clock transition at output register feedback, from output pin to any register input. $1/(t_{SU} + t_{CO})$

f_{MT}: F_{max} without feedback; maximum clock frequency with no register feedback or inputs. Maximum register toggle rate. Determined with minimum clock period. $1/(t_{CH} + t_{CL})$

Absolute Maximum Ratings (Note 1)

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/Distributors for availability and specifications.

Supply Voltage (V_{CC})	−0.5V to +7.0V
Input Voltage (Note 2)	−2.5V to V_{CC} +1.0V
Off-State Output Voltage (Note 2)	−2.5V to V_{CC} + 1.0V
Output Current	±100 mA
Storage Temperature	−65°C to +150°C
Ambient Temperature with Power Applied	−65°C to +125°C
Junction Temperature	−65°C to +150°C
Lead Temperature (Soldering, 10 seconds)	260°C
ESD Tolerance	500V

C_{ZAP} = 100 pF
R_{ZAP} = 150Ω
Test Method: Human Body Model
Test Specification: NSC SOP-5-028 REV.C

Recommended Operating Conditions

SUPPLY VOLTAGE AND TEMPERATURE

Symbol	Parameter	Commercial			Industrial			Military			Units
		Min	Typ	Max	Min	Typ	Max	Min	Typ	Max	
V_{CC}	Supply Voltage	4.75	5	5.25	4.5	5	5.5	4.5	5	5.5	V
T_A	Operating Free-Air Temperature	0	25	75	−40	25	85	−55	25		°C
T_C	Operating Case Temperature									125	°C

Electrical Characteristics Over Recommended Operating Conditions

Symbol	Parameter	Conditions		Temperature Range	Min	Typ	Max	Units
V_{IH}	High Level Input Voltage				2.0		V_{CC}+1	V
V_{IL}	Low Level Input Voltage				−1.0		0.8	V
V_{OH}	High Level Output Voltage	V_{CC} = Min	I_{OH} = −3.2 mA	COMM/IND	2.4			V
			I_{OH} = −2.0 mA	MIL	2.4			V
V_{OL}	Low Level Output Voltage	V_{CC} = Min	I_{OL} = 8 mA	COM/IND			0.5	V
			I_{OL} = 8 mA	MIL			0.5	V
I_{OZH}	High Level Off State Output Current	V_{CC} = Max, V_O = V_{CC} (Max)					10	μA
I_{OZL}	Low Level Off State Output Current	V_{CC} = Max, V_O = GND					−10	μA
I_I	Maximum Input Current	V_{CC} = Max, V_I = V_{CC} (Max)					10	μA
I_{IH}	High Level Input Current	V_{CC} = Max, V_I = V_{CC} (Max)					10	μA
I_{IL}	Low Level Input Current	V_{CC} = Max, V_I = GND					−10	μA
I_{OS}*	Output Short Circuit Current	V_{CC} = 5.0V, V_O = GND			−30		−160	mA
I_{CC}	Supply Current	f = 25 MHz, V_{CC} = Max		COM			140	mA
				MIL/IND			160	mA
C_I	Input Capacitance	V_{CC} = 5.0V, V_I = 2.0V					10	pF
$C_{I/O}$	I/O Capacitance	V_{CC} = 5.0V, $V_{I/O}$ = 2.0V					12	pF

***Note:** One output at a time for a maximum duration of one second.

Note 1: Absolute maximum ratings are those values beyond which the device may be permanently damaged. Proper operation is not guaranteed outside specified recommended operating conditions.

Note 2: Some device pins may be raised above these limits during programming and preload operations according to the applicable specification.

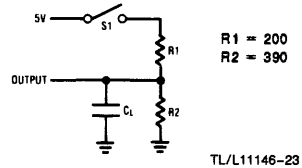
Switching Characteristics Over Recommended Operating Conditions

AC TIMING REQUIREMENTS

Symbol	Parameter		Conditions	MAPL128-45 COM		MAPL128-40 COM		MAPL128-33 COM		Units
				Min	Max	Min	Max	Min	Max	
t_{SU}	Set-Up Time (Input or Feedback before Clock)			17		17		20		ns
t_H	Hold Time (Input after Clock)			0		0		0		ns
t_W	Clock Pulse Width (High/Low)			8		8		8		ns
t_{CYCLE}	Clock Cycle Period (with Buried Feedback)			22		25		30		ns
f_{MB}	Clock Frequency	with Buried Feedback*			45.5		40.0		33.3	MHz
f_{MO}		with I/O Feedback*			40.0		40.0		33.3	MHz
f_{MT}		without Feedback			62.5		62.5		62.5	MHz
t_{PR}	Clock Valid after Power-Up				100		100		100	ns
t_{CLK}	Clock Input to Registered Output or Feedback		S1 Closed, C = 50 pF		8		9		10	ns
t_{PZXG}	$\overline{OE}$ ↓ to Registered Output Enabled		Active High; S1 Open, C_L = 50 pF Active Low; S1 Closed, C_L = 50 pF		10		10		12	ns
t_{PXZG}	$\overline{OE}$ ↑ to Registered Output Disabled		From V_{OL}; S1 Open, C_L = 5 pF From V_{OH}; S1 Closed, C_L = 5 pF		10		10		12	ns
t_{PZXI}	Input to Output Enable via Product Term		Active High; S1 Open, C_L = 50 pF Active Low; S1 Closed, C_L = 50 pF		15		15		18	ns
t_{PXZI}	Input to Output Disabled via Product Term		From V_{OL}; S1 Open, C_L = 5 pF From V_{OH}; S1 Closed, C_L = 5 pF		15		15		18	ns
t_{RESET}			S1 Closed, C = 50 pF		45		45		45	μs
t_{IPD}					20		20		20	ns
t_{IRC}					20		20		20	ns

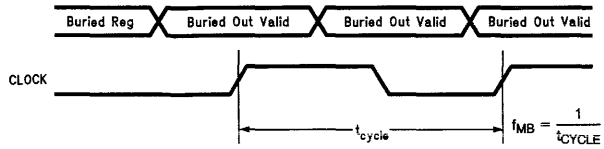
*Actual measurements

AC Test Load



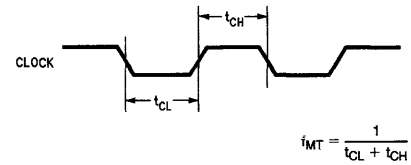
TL/L11146–23

Timing Diagrams



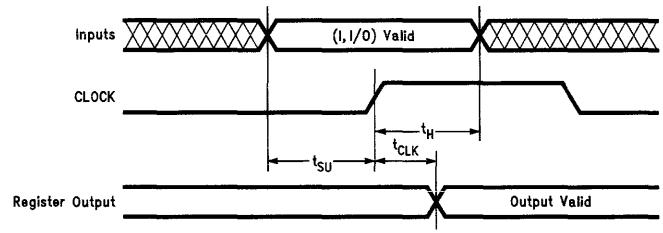
TL/L/11146–12

FIGURE 8. Internal Feedback Frequency



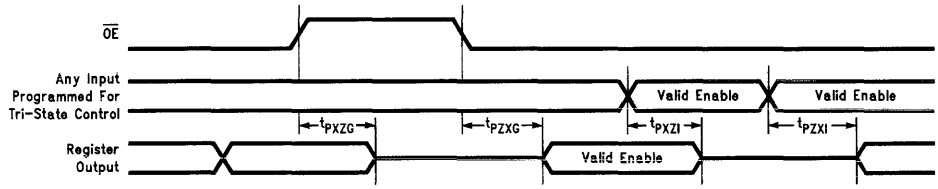
TL/L/11146–13

FIGURE 9. Toggle Frequency



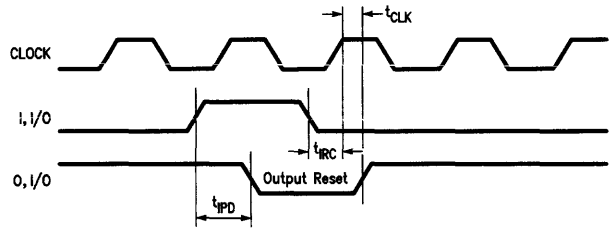
TL/L/11146–14

FIGURE 10. External Feedback



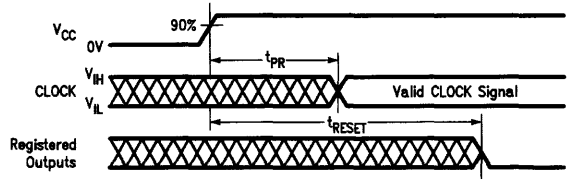
TL/L/11146–15

FIGURE 11. Output Enable Timing



TL/L/11146–16

FIGURE 12. Initialization Timing



TL/L/11146–17

FIGURE 13. Power-Up Reset Timing

PLD Comparisons

The multiple PLA architecture used by the MAPL product family can be configured to implement a number of different architectures commonly used in systems today. In sequential applications MAPL can superset the following existing PLD architectures:

— Field Programmable Logic Sequencers (FPLS)
— Registered PAL
— Programmable Microsequencers/Micro Controllers
— Low Density Gate Arrays/Logic Cell Arrays (LCA)
— Registered PROM

The MAPL architecture optimally implements the FPLA or FPLS architecture, since it has both a programmable AND array and a programmable OR array. The MAPL128 core offers competitive performance and significant density and power advantages over these other FPLA type products.

The devices can also emulate bit-slice microcode sequencer instructions and programmable microsequencer with buried register stack operations. MAPL offers superior branch logic capability over existing products.

The MAPL architecture can also perform a reduced set of microcontroller applications. These functions and other system/bus specific functions are often implemented in low density gate arrays.

By programming the AND array into a fixed decode function, a wide word registered PROM architecture with buried registers can be implemented. In order to take full advantage of the architecture in this application, a portion of the data word must be dedicated to generation of the next address.

Design Technology Testability

Electrically Erasable (EE)CMOS technology allows the MAPL product family to be 100% AC, DC and functionally tested prior to shipment. Unlike one-time programmable devices, the actual array's true worst case patterns are programmed into the device and tested over temperature and voltage ratings. The specifications are guaranteed to be true worst case parameters, with fully loaded internal arrays and all (maximum available) outputs switching. The erasable cells allow for the device to be programmed and functionally tested, providing the highest programming yields and post-programming, functional yields, available in a user programmable device. National Semiconductor guarantees 100% field programmability of MAPL products.

Register Preload

All macrocell registers have the ability to be preloaded and functionally tested. This capability allows the device to be forced into undefined or arbitrary states to greatly reduce the required number sequential test vectors. The device may be put into any desired register state at any point during the functional test sequence. This allows complete verification of sequential logic circuits including states that are not defined in the normal operational state sequence. The register preload is not an operational mode. It is available in certified programming equipment for test purposes. The register preload algorithm for the MAPL product family, is available separately from National Semiconductor.

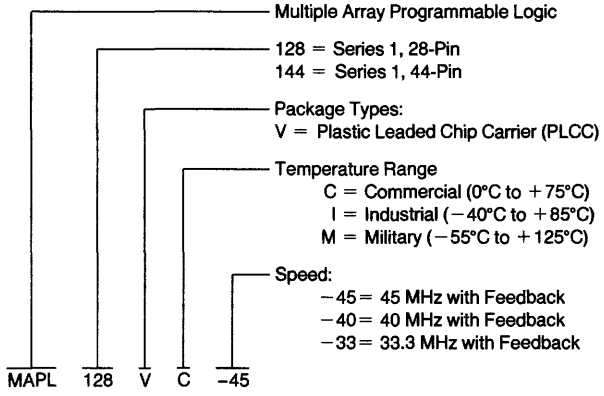
Electronic Signature

Each MAPL product contains 16 bytes of user reprogrammable memory known as the User Electronic Signature (UES). The UES may contain any identification information desired by the user. This information typically is associated with revision numbers, dates, inventory control information, identification data and names. The information is read out of the device using the programming equipment's verification procedure. OPAL software allows representation of the UES data in HEX, binary or ASCII formats.

Design Security

A security cell is provided on all MAPL products as a deterrent to copying the logic design. If the security cell is programmed, programming and verification of the array is disabled. Once the cell is programmed, only a "bulk erase" of the whole product will erase the security cell, thus the original design configuration can never be examined.

Ordering Information



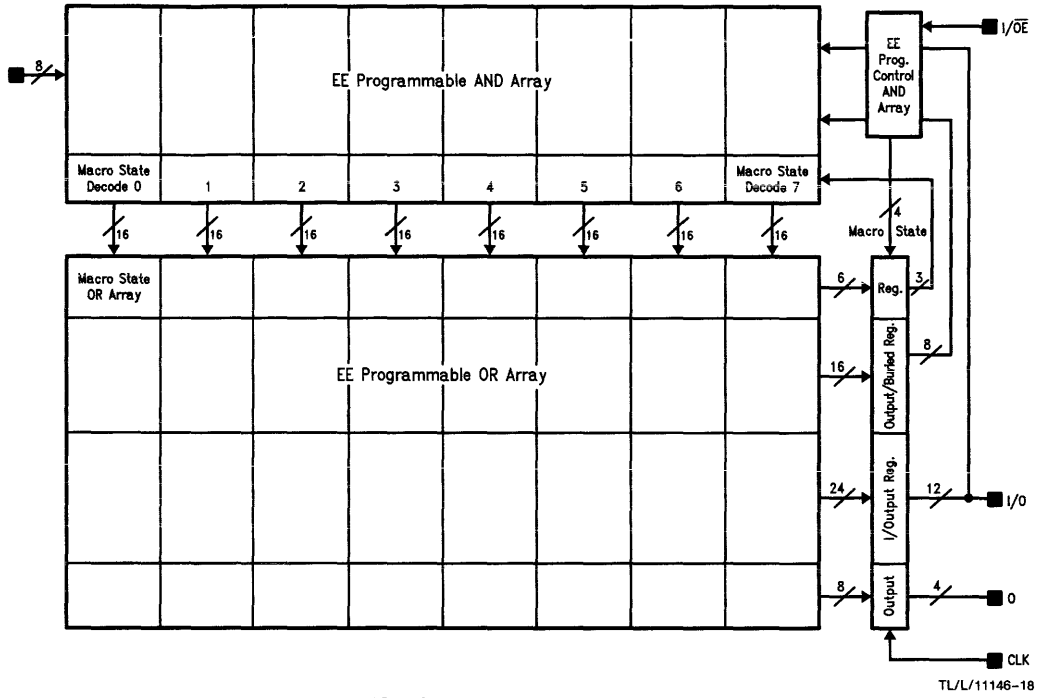


FIGURE 14. MAPL128 Block Diagram

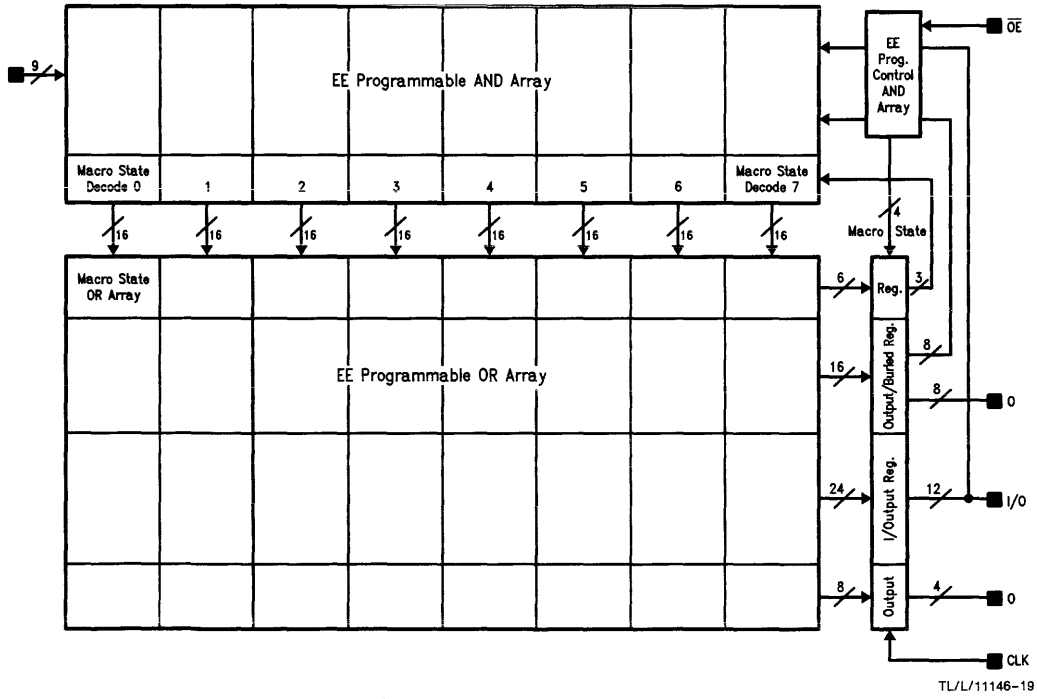
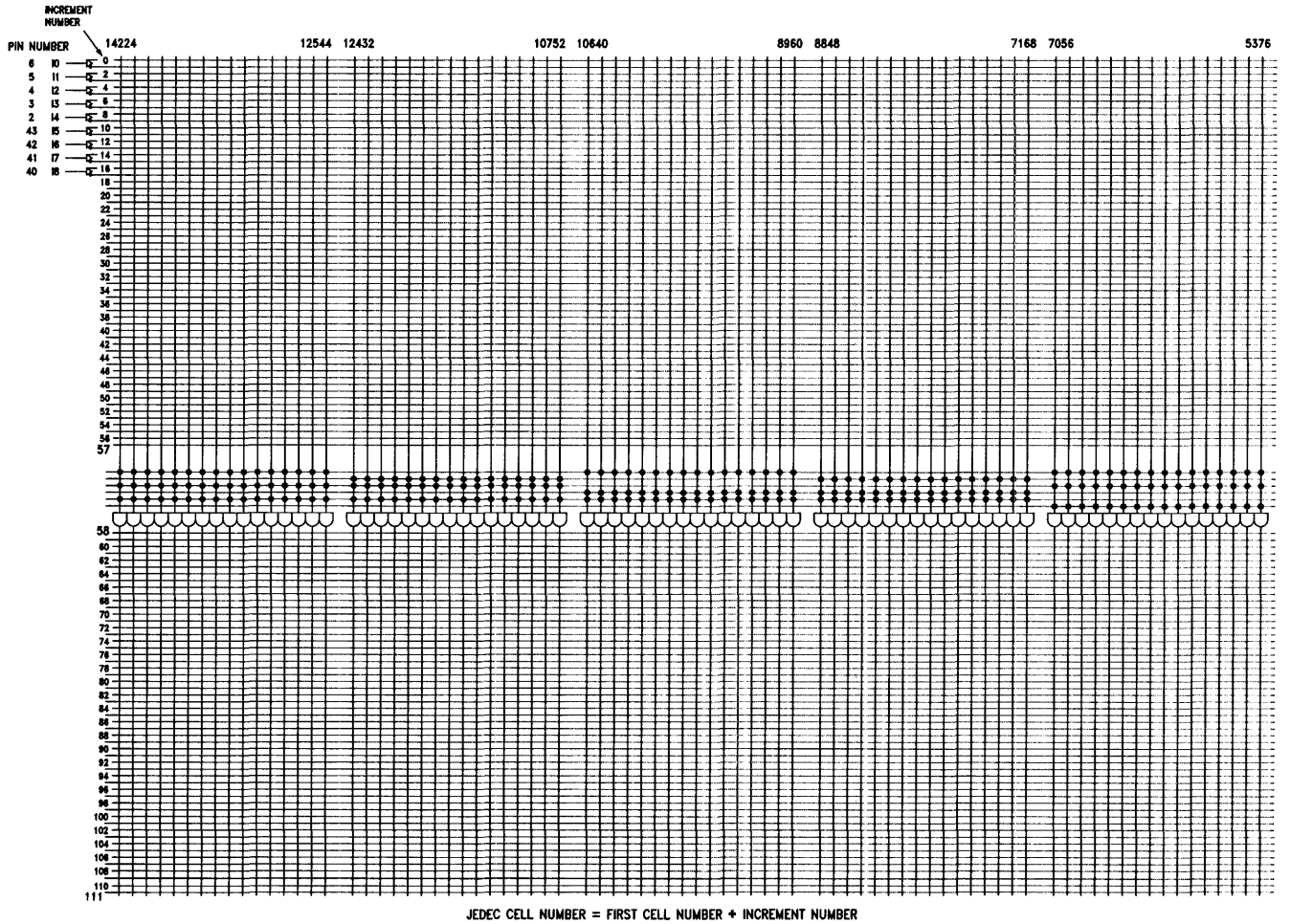
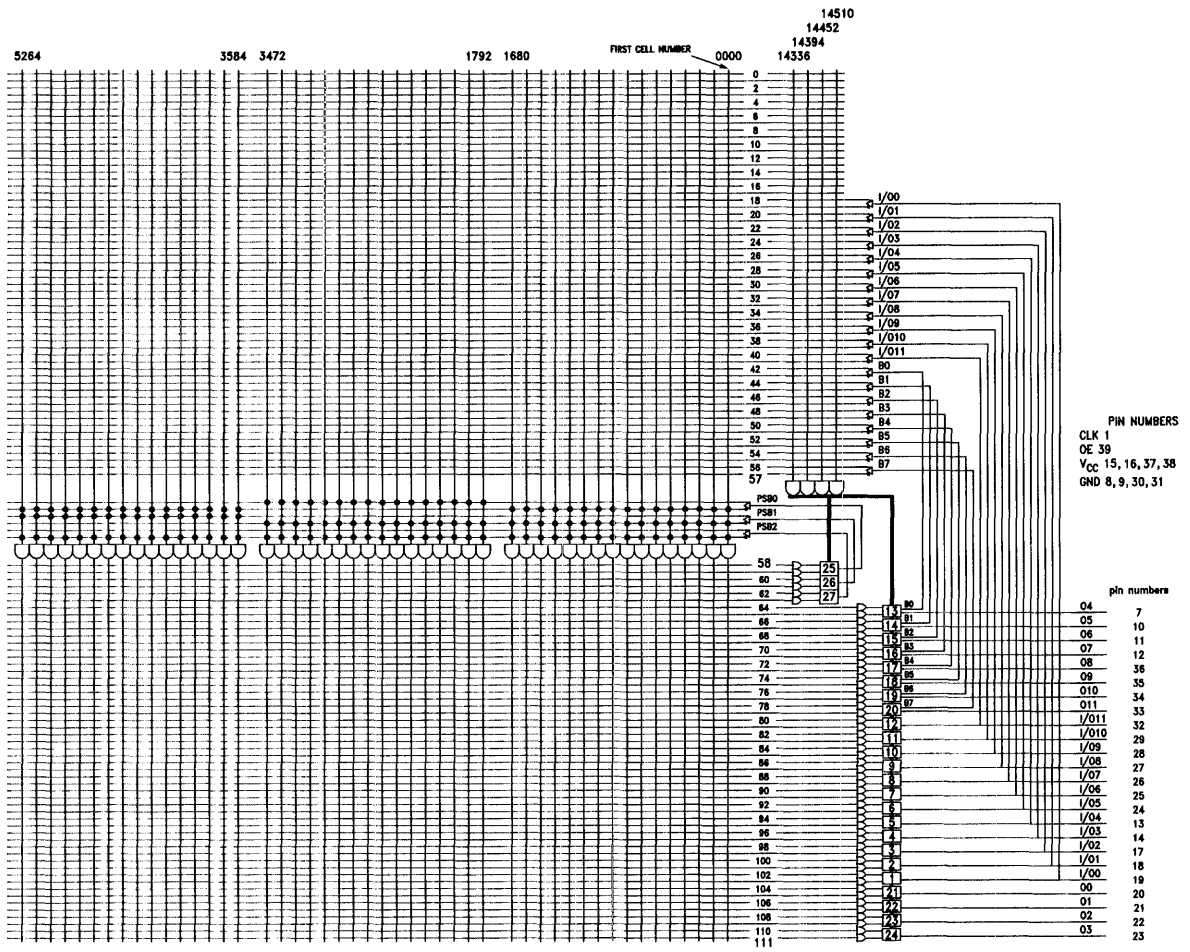


FIGURE 15. MAPL144 Block Diagram



TL/L/11146–20

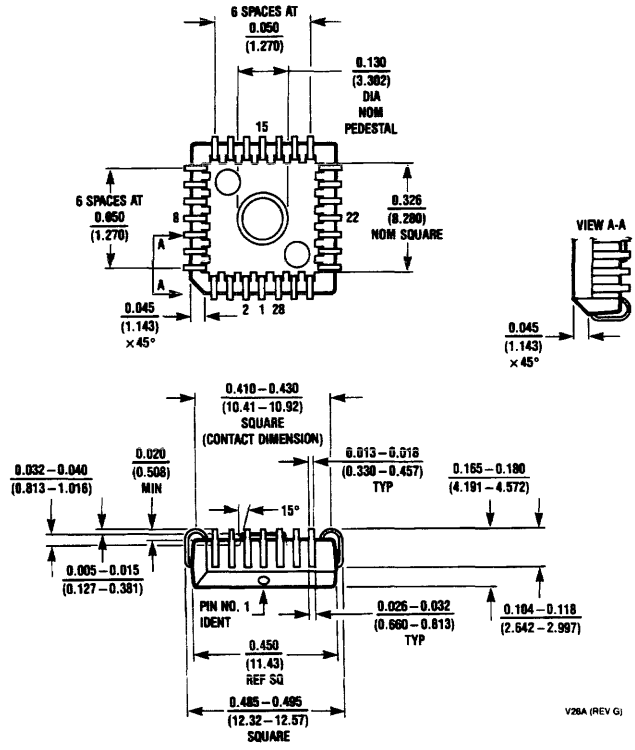
FIGURE 16. MAPL144 Logic Diagram



TL/L/11146–21

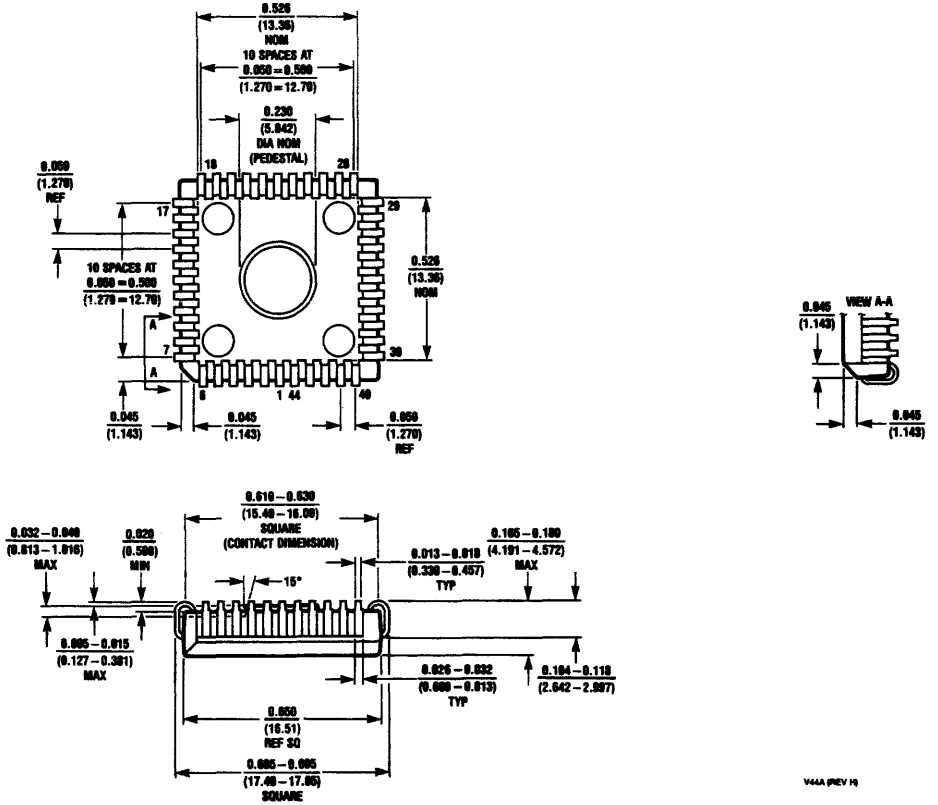
FIGURE 16. MAPL144 Logic Diagram (Continued)

Physical Dimensions inches (millimeters)



28-Lead Plastic Chip Carrier (V)
Order Number MAPL128V
NS Package Number V28A

Physical Dimensions inches (millimeters) (Continued)



44-Lead Plastic Chip Carrier (V)
Order Number MAPL144V
NS Package Number V44A

LIFE SUPPORT POLICY

NATIONAL'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS WRITTEN APPROVAL OF THE PRESIDENT OF NATIONAL SEMICONDUCTOR CORPORATION. As used herein:

1. Life support devices or systems are devices or systems which, (a) are intended for surgical implant into the body, or (b) support or sustain life, and whose failure to perform, when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in a significant injury to the user.
2. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

National Semiconductor Corporation
2900 Semiconductor Drive
P.O. Box 58090
Santa Clara, CA 95052-8090
Tel: 1(800) 272-9959
TWX: (910) 339-9240

National Semiconductor GmbH
Industriestrasse 10
D-8080 Furstenfeldbruck
West Germany
Tel: (0-81-41) 103-0
Telex: 527-649
Fax: (08141) 103554

National Semiconductor Japan Ltd.
Sanseido Bldg. 5F
4-15 Nishi Shinjuku
Shinjuku-Ku,
Tokyo 160, Japan
Tel: 3-299-7001
FAX: 3-299-7000

National Semiconductor Hong Kong Ltd.
Suite 513, 5th Floor
Chinachem Golden Plaza,
77 Mody Road, Tsimshatsui East,
Kowloon, Hong Kong
Tel: 3-7231290
Telex: 52996 NSSEA HX
Fax: 3-3112536

National Semiconductores Do Brasil Ltda.
Av. Brig. Faria Lima, 1383
6.0 Andor-Conj. 62
01451 Sao Paulo, SP, Brasil
Tel: (55/11) 212-5066
Fax: (55/11) 211-1181 NSBR BR

National Semiconductor (Australia) PTY, Ltd.
1st Floor, 441 St. Kilda Rd.
Melbourne, 3004
Victory, Australia
Tel: (03) 267-5000
Fax: 61-3-2677458

CHAPTER 2
MAPL244/268 ADVANCE INFORMATION

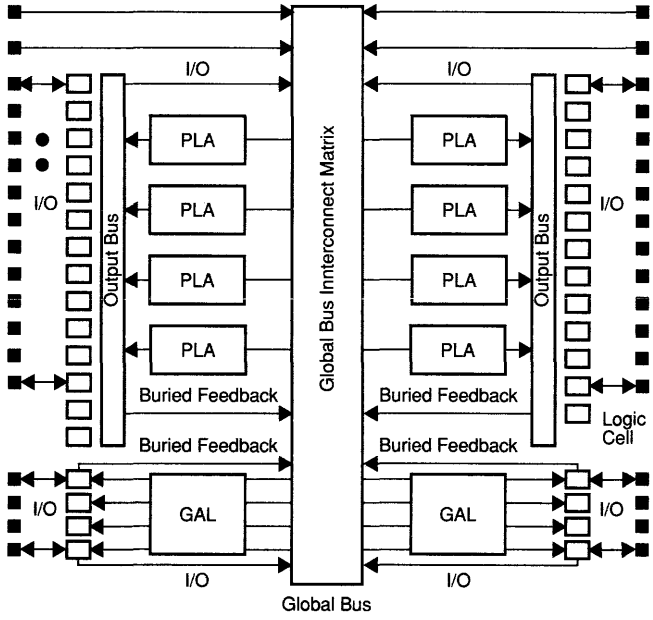
2.1 Introduction

The MAPL244 and MAPL268 both integrate FPLA and PAL architectures, which make it suitable for large sequential and combinatorial applications. The FPLA is similar to the MAPL28, thus allowing sequential applications, while the PAL architecture is like the 22V10, thus giving also combinatorial capabilities to the product.

A simple programmable multiplexing network allows for internal connection between the FPLA and the PAL and, also, gives some flexibility in selecting the PAL and the FPLA inputs. The FPLA module can receive, as inputs, some of the FPLA registers, page registers, and all the I/O pins. The PAL module can receive as inputs the FPLA buried registers, page registers and almost all the I/O pins.

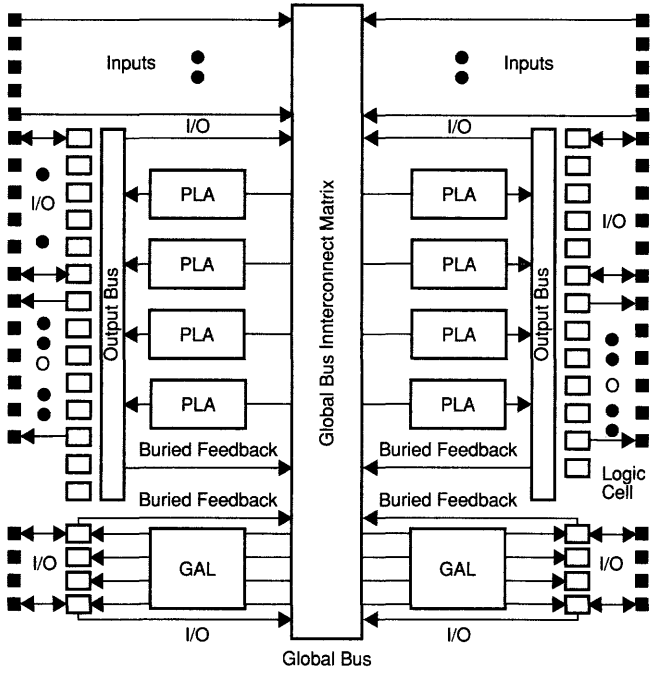
The PAL module is non-paged and always remains active. It is able to achieve higher performance than the FPLA module because it is programmable only in the AND plane. The PAL module has a variable type OLMC like the 22V10 with the enhancements of dynamic polarity select and asynchronous clock option.

Both the FPLA and PAL have additional product terms for tri-state control and register initialization and asynchronous set/reset. The PAL has also additional product terms for implementing an asynchronous clock and dynamic polarity bit for each output. Both these features are an enhancement to the 22V10 architecture.



TSP-MAPL-01

Figure 2-1 MAPL244 Block Diagram



TSP-MAPL-02

Figure 2-2 MAPL268 Block Diagram

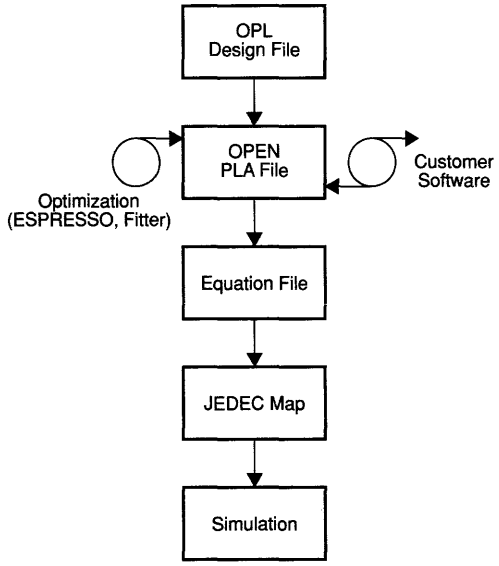
CHAPTER 3
OPAL PLD DEVELOPMENT SOFTWARE

3.1 Introduction

National Semiconductor Corporation has developed a new Open Programmable Architecture Language (OPAL™) software package that allows system designs to implement custom circuit functions based on OPEN PLA software formats. OPAL allows the user to interface with his existing design tools, third party software tools, ASIC design tools and public domain development tools. The interface is accomplished with an extended PLA file format. OPAL accepts all major design syntaxes and runs under DOS, UNIX, and VMS operating systems. The OPAL design input format compiles Boolean equations, truth tables and state machine language descriptions in the same file. This input file (.OPL) is device and architecture independent. National's OPAL software allows designers to compile into high density programmable devices with complete design verification, translation and observation at any point in the design flow.

The OPAL design file is device independent and may include any or all of the functional blocks, state machine language, multiple level Boolean equations and truth table formats. The design file is parsed and compiled into a PLA file. The PAL file can be fitted, partitioned or minimized into a MAPL™ product or standard PAL/GAL devices. The PLA file is translated into "flattened" Boolean equations. The flattened equations file is user compiled into a JEDEC map.

Partitioning the software into these modules allows designers to observe and verify designs at any point in the design flow. The OPAL software package offers a designer the ideal set of tools required for system design.



TSP-MAPL-03

Figure 3-1 NSC OPAL PLD Development Software

OPAL software consists of a series of modules that accept one format as input and create another format as output. Files created by OPAL use file extensions to describe the format. For example, the OPL2PLA module requires an OPAL file, "*.OPL", and generates a PLA file, "*.PLA". The names of the modules are easy to remember because they describe the action the module takes.

OPAL modules includes:

1. OPL2PLA
2. PLA2EQN
3. EQN2JED
4. JED2EQN
5. EQN2OPL
6. *FITMAPL
7. *ESPRESSO
8. OPALSIM

By simply typing the module name alone, all OPAL modules present a screen of commands. The exception to this, however, is the ESPRESSO module.

* The modules FITMAPL and ESPRESSO both take a PLA file as input and make a PLA file as output.

A sample design flow fora MAPL product on "test.opl" is

OPL2PLA test	(produces test.pla)
FITMAPL test -m	(fits test.pla into MAPL128 with minimization)
PLA2EQN test.out	(makes test.eqn from output of FITMAPL run)
EQN2JED test	(compiles test.eqn into a JEDEC map)

For non-MAPL parts, to use minimization on test1.opl use

OPL2PLA test1	(makes test1.pla)
ESPRESSO test1.pla	(makes test1.out from test1.pla)
PLA2EQN test1.out	(converts to boolean equations)
EQN2JED test1	(makes test1.jed from test1.eqn)

3.2 OPAL Software Modules

OPL2PLA

OPL2PLA compiles an OPAL source file and produces an OPEN-PLA file as output. OPAL is a hardware description language for specifying logic designs. It can be used to specify logic in terms of state machines, truth tables, or Boolean equations. The OPAL syntax is fully described in the "Language Reference" section of this manual.

PLA2EQN

The PLA2EQN module translates an open PLA format file to a standard sum-of-products (SOP) boolean equation file. The PLA file used as input to this module can be either:

1. The output of the OPL2PLA module, or

2. The output of the ESPRESSO minimization program, or
3. The output of the fitter, or
4. The output of a third-party package, or
5. A text file that matches the PLA file format.

EQN2JED

EQN2JED is a standard SOP Boolean equations to JEDEC file assembler for programmable logic devices (PLDs). One of the many features of this module is that it has automatic pin list generation.

The JEDEC file contains all the necessary design details which can be downloaded to a device programmer for programming the target PLD. The JEDEC file is fully compatible with JEDEC standard 3A which is supported by the industry-standard device programmers.

JED2EQN

JED2EQN will disassemble a JEDEC file into the corresponding boolean equations.

The labels used in the boolean equations created by JED2EQN contain the pin number preceded by the type of signal. Observing the labels makes it easy to determine if the pin is used as a dedicated input, combinatorial or registered output, and whether or not the output is used as a feedback into the device.

EQN2OPL

EQN2OPL converts Boolean equations specified in EQN format to Boolean equations in OPAL format.

The utility program file preserves all the comments available in the equation file. The pin declarations will be preserved but commented out in the output OPAL file.

PAL2GAL

PAL2GAL converts a PAL JEDEC file into a GAL JEDEC file. PAL2GAL first checks to ensure that the PAL is replaceable by a GAL before it proceeds to do the conversion.

FITMAPL

FITMAPL fits a PLA file into any of the MAPL devices. The process is accomplished by assigning pin/node numbers to the pin/node labels. Two products are supported by this module, namely the MAPL128 and MAPL144.

ESPRESSO

ESPRESSO minimizes the logic in a PLA file and writes out a new PLA file. It is a public domain utility from UC Berkeley.

OPALSIM

OPALSIM is a simple, but powerful logic simulator, that enables the designer to analyze digital circuits using a personal computer.

Two files are needed to run OPALSIM. These two files consists of a macrofile [.MAC] and a circuit file [.CKT]. The macro file is obtained by running the JEDEC file through the conversion utility MAPL2MAC. The circuit file is user-defined with test vectors and miscellaneous simulator commands (i.e., probes, timing, etc.). Once these files are in place, the user can simulate a design using OPALSIM and view the output waveforms using OPALVIEW.

3.3 Integrating Multiple Functions into a High Density PLD Using Efficient Mapping Algorithms

Introduction

Upon conception, PLDs were used primarily for integrating multiple TTL parts into one package. Now as we enter a new era of programmable logic, high density PLDs are beginning to integrate multiple medium size PAL and GAL devices as well as discrete logic into one device.The MAPL family offers the ability to integrate multiple functions with a reduction in board space, power consumption, and cost. The MAPL architecture consists of multiple PLAs that are interconnected via programmable macrostate registers and a global input bus. This architecture allows for easy implementation of complex state machines, sequencers, controllers, and general synchronous logic designs.

The integration of multiple functions can be accomplished with two different types of partitioning: I/O partitioning and logic partitioning. The basic difference between the two is that I/O partitioning is external partitioning, whereas logic partitioning can be thought of as internal to the device for some high density PLD's such as the MAPL. For example, I/O partitioning would involve separating a 32-input PLA into four 8-input PLAs to fit into four PAL type devices. On the other hand, logic partitioning involves separating the logic in the PLA table to fit in separate arrays. This section will focus on logic partitioning.

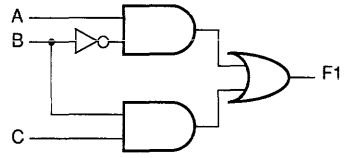
Logic Partitioning

What makes the MAPL family unique is its proprietary architecture which dynamically allocates user-defined product and sum terms based on macrostate assignment. This partitioning architecture solves the problems associated with performance, power consumption, and product term limitation seen with traditional PAL and GAL devices. By determining the input or state dependency required for each logic term, a design can be partitioned so that only the required logic terms are active. Thus, only a portion of the device is powered up at any given time. This implementation utilizes three global macrostate registers which are driven by the same product terms required for output and state transitions. These three macrostate registers then enable the next portion of the array with no performance penalty.

OPEN PLA Format

The logic partitioning and PLA allocation can be accomplished, automatically or manually, by using the user-defined global macrostate feedback registers. The MAPL series is supported by National's OPAL software package and by other third party software packages that support the OPEN PLA format. The three figures below will help to clarify the OPEN PLA format. Figure 3-2 is a basic schematic. Figure 3-3 is the schematic represented as a Boolean equation. Figure 3-4 is the same schematic represented in the OPEN PLA format. This figure shows the basic features of the PLA format. The first four lines of the PLA are the header information. .i and.o list the number of inputs and outputs respectively. .ilb and .ob are the input and output labels. This file is a truth table that describes one equation (F1) that has 3 inputs (A, B, and C). The input

side of the table (first three columns) uses a "1", "0", or a "-" for a true, complement, or don't care, respectively. A don't care can be considered as a no connect. The output side of the table (last column) uses a "1" to indicate that the input row (product term) is used and a "0" to indicate the row is not used. The OPEN PLA format is the simplest way to represent logic so that it can be partitioned and minimized. The "OPEN" in OPEN PLA means the user has access to this file. Thus, the user can manually revise files at any time during the design process. This can be useful in designs with partitioning and implementation issues. OPAL brings manual and automatic partitioning into one design environment.



TSP-MAPL-04

Figure 3-2 Schematic

```
F1 = (B & C) # (A & !B);
```

Figure 3-3 Boolean Equation

```
.i 3
.o 1
.ilb A B C
.ob F1
-11 1
10- 1
.e
```

Figure 3-4 Open PLA Format

Types of Logic Partitioning

OPAL contains a "fitter" program which automatically allocates logic to arrays within the device. The first two algorithms that will be explained below are automated processes. The third algorithm listed below is a manual partitioning algorithm which can be used with OPAL's OPEN PLA format. Many of the proprietary software

packages on the market use binary files to represent their intermediate files. In other words, these packages do not let the user manually edit or view intermediate files. OPAL, on the other hand, allows the user to get into the partitioning process. That way, users know exactly what they are getting and can alter the output to suit their needs.

Sequential Partitioning

One of the initial partitioning algorithms used for devices such as the MAPL series was sequential partitioning. This type of partitioning first sorts the terms with respect to the present state. It then places these terms into the device in an incremental order. When the logic in the first array is utilized, it jumps to the next array and continues to place terms in the succeeding array and so on. For instance, if the PLA table in Figure 3-5 was the input to a sequential partitioning algorithm like the one described above, the output would resemble the PLA table in Figure 3-6. Notice how the algorithm also appends three terms to the input and output columns for activating the appropriate logic array.

In Fgure 3-6, besides appending the three terms to the input and output columns, the algorithm also takes care of transitions from array to array. Notice the present and next state of the array bits in the last line of each array in Fgure 3-6. This algorithm is not very efficient in that it does not minimize while it partitions a PLA table. Some more complicated designs may require minimization.

State Dependency Partitioning

State dependent partitioning soon followed with the need for minimization along with partitioning. The state dependent algorithm reads through the PLA and sorts the state and feedback bits. It then selects a target bit with which to split the logic into different arrays. This is the automated partitioning algorithm which OPAL uses in its design environment. Compared to sequential partitioning, this algorithm is more intelligent when it comes to partitioning because it also involves minimization.

Multiple Level Logic Partitioning

The idea of multiple level logic partitioning was conceived when the previously described state dependency algorithm would fail to fit a PLA table into a given device. The multiple level logic parti-

```
.i 10
.o 11
.type f
.phase 11111111111
.ilb advance delay 13 12 11 10 d2 d1 d0 a0
.ob cancel_adv.reg cancel_del.reg fsr.reg
   13.reg 12.reg 11.reg 10.reg d2.reg d1.reg
   d0.reg a0.reg
100000---0       1-00110---1
100010---0       1-01000---1
100011---0       1-01001---1
100100---0       1-01-10---1
100101---0       1-01011---1
100110---0       1-01100---1
100111---0       1-01101---1
101000---0       1-01110---1
101001---0       1-01111---1
101010---0       1-01-11---1
101011---0       0-01100---0
101100---0       0-01101---0
101101---0       0-01110---0
101110---0       0-01111---0
101111---0       0-00000---0
01--------       -------000-
01----000-       -00----001-
01----001-       -00----011-
01----011-       -00----000-
.e
```

Figure 3-5 Input PLA Table

```
.i 13
.o 14
.type f
.phase 11111111111
.ilb advance delay 13 12 11 10 d2 d1 d0 a0 ar2 ar1 ar0
.ob cancel_adv.reg cancel_del.reg fsr.reg
   13.reg 12.reg 11.reg 10.reg d2.reg d1.reg
   d0.reg a0.reg ar2.reg ar1.reg ar0.reg
100000---0000    1-00110---1000
100010---0000    1-01000---1000
100011---0000    1-01001---1000
100100---0000    1-01-10---1000
100101---0000    1-01011---1000
100110---0000    1-01100---1000
100111---0000    1-01101---1000
101000---0000    1-01110---1000
101001---0000    1-01111---1000
101010---0000    1-01-11---1000
101011---0000    0-01100---0000
101100---0000    0-01101---0000
101101---0000    0-01110---0000
101110---0000    0-01111---0000
101111---0000    0-00000---0000
01--------000    ----------001
01--------001    -------000-001
01----000-001    -00----001-001
01----001-001    -00----011-001
01----011-001    -00----000-001
10--------001    ----------000
.e
```

Array #1

Array #2

Figure 3-6 Sequential Partitioning Output PLA Table

tioning algorithm is a manual process that introduces an additional level of logic by which to partition. The reason the state dependency algorithm fails is not always due to exceeding the number of product terms allowed. As mentioned earlier, if the target bit selected has many "don't-cares", these terms will get expanded with the target bit overwritten with a "zero" and "one". These terms then get minimized but if this process needs to be done more than once, it increases the chance of the PLA table not fitting into the device. This is where the multiple level logic partitioning algorithm comes into play.

The primary function of the multiple level logic partitioning algorithm is to add an XLT (eXtra Logic Term) to the PLA table, which in turn can be assigned as the state dependent control of different arrays. The steps involved in this algorithm are as follows:

1. If the state dependent algorithm is unable to partition the PLA table, separate the logic into functions that can be evaluated at different times.

2. Assign an XLT, eXtra Logic Term, feedback. For example, if the PLA can be divided into two separate functions, the XLT will be a "1" for one function and a "0" for the other.

3. Run the revised PLA table with the XLT through the state dependent algorithm.

Figure 3-7 is an example of a PLA table that will fail to fit using the state dependency algorithm by itself. Notice how most of the feedback terms in the input column are "don't cares". This is the primary reason why this particular PLA table will not fit into the device. The solution is shown in Figure 3-8. An XLT is added to the input and output column of the PLA table. The term added has all the makings of being a rather close ideal target bit - even distribution of "ones" and "zeros" and no "don't-cares". The PLA table in Figure 3-8 will now fit into the device using the state dependency algorithm. The output is shown in Figure 3-9.

Now, it looks all plain and simple, but once a term is added to allow moving from one array to another, it adds a level of logic and thus a time delay to the circuit. This is where the designer has to be careful. As a rule of thumb when adding the extra feedback term, separate the logic with respect to the time it is evaluated. If an output does not need to be evaluated at the same time as other

```
.i 10
.o 11
.type f
.phase 11111111111
.ilb advance delay 13 12 11 10 d2 d1 d0 a0 ar2 ar1 ar0
.ob cancel_adv.reg cancel_del.reg fsr.reg
   13.reg 12.reg 11.reg 10.reg d2.reg d1.reg
   d0.reg a0.reg
10--------       1-00110---1
10----1---       1-01000---1
10-11-----       1-01001---1
10-1------       1-01010---1
10-1-1----       1-01011---1
10--1-----       1-01100---1
10---1----       1-01101---1
111---1--0       1-01110---1
11-1---1--       1-01111---1
11--1---10       1-01011---1
11---11---       0-01100---0
110------0       0-01101---0
11-0----1-       0-01110---0
11--0-1--0       0-01111---0
11---0-1--       0-00000---0
01--------       -------000-
01------0-       -00----001-
01------1-       -00----011-
01-----11-       -00----000-
.e
```

Figure 3-7 Input PLA Table

```
.i 11
.o 12
.type f
.phase 11111111111
.ilb advance delay 13 12 11 10 d2 d1 d0 a0 ar2 ar1 ar0
.ob cancel_adv.reg cancel_del.reg fsr.reg
   13.reg 12.reg 11.reg 10.reg d2.reg d1.reg
   d0.reg a0.reg XLT.reg
10--------0      1-00110---10
10----1---0      1-01000---10
10-11-----0      1-01001---10
10-1------0      1-01010---10
10-1-1----0      1-01011---10
10--1-----0      1-01100---10
10---1----0      1-01101---10
111---1--00      1-01110---10
11-1---1--0      1-01111---10
11--1---100      1-01011---10
11---11---0      0-01100---00
110------00      0-01101---00
11-0----1-0      0-01110---00
11--0-1--00      0-01111---00
11---0-1--0      0-00000---00
01--------0      -------000-0
01------0-0      -00----001-0
01------1-0      -00----011-0
01-----11-0      -00----000-0
.e
```

Figure 3-8 Input PLA Table with Added XLT

```
          .i 14
          .o 15
          .type f
          .phase 11111111111
          .ilb advance delay 13 12 11 10 d2 d1 d0 a0 XLT ar2 ar1 ar0
          .ob cancel_adv.reg cancel_del.reg fsr.reg
              13.reg 12.reg 11.reg 10.reg d2.reg d1.reg
_________     d0.reg a0.reg XLT.reg ar2.reg ar1.reg ar0.reg
          10--------0000   1-00110---10000
          10----1---0000   1-01000---10000
          10--11----0000   1-01--1---10000
          10-1------0000   1-01010---10000
          10-1-1----0000   1-01011---10000
          10--1-----0000   1-01100---10000
Array #1  10---1----0000   1-01101---10000
          11------------   -----------1001
          01--------0000   -------000-0000
          01------0-0000   -00----001-0000
          01------1-0000   -00----011-0000
_________ 01-----11-0000   -00----000-0000
          110------01001   0-01101---00000
          11-0----1-1001   0-01110---00000
          11--0-1--01001   0-01111---00000
          11---0-1--1001   0-00000---00000
Array #2  111---1--01001   1-01110---10000
          11-1---1--1001   1-01111---10000
          11--1---101001   1-01011---10000
_________ 11---11---1001   0-01100---00000
          .e
```

Figure 3-9 Multiple Level Logic Partitioning Output PLA Table

outputs, this term can be placed in a different array. The designer just has to remember that there is a transition from the added level of logic, which is the XLT, to the normal level of logic. Most applications do not have all the outputs evaluated at the same time, and therefore, the multiple level logic partitioning algorithm can be a very helpful process.

Conclusion

A variety of partitioning algorithms have been presented. Depending on the complexity of the design and the needs of the designer, one of these variations should prove helpful. The MAPL devices, as well as the OPAL software, have been successful in integrating functions for many internal applications.

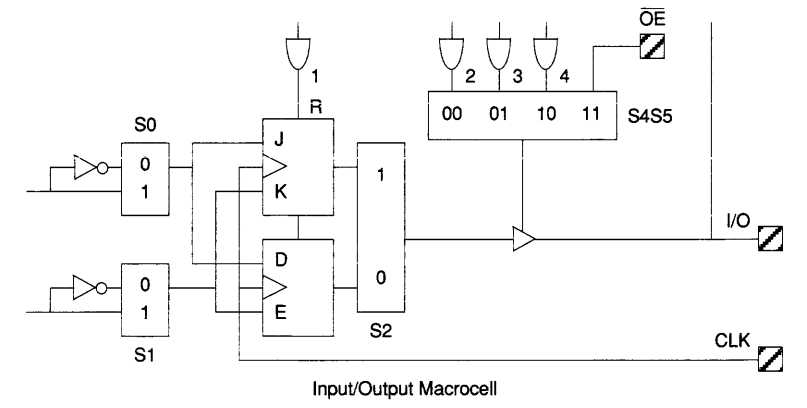
CHAPTER 4

APPLICATION EXAMPLES

4.1 MAPL Register Control

The following application demonstrates the use of OPAL to control any single line in the output macrocell of the MAPL128 and configure the output as either a JK or DE register. The example also shows how to drive the reset and the tri-state functions. From the I/O macrocell shown in Figure 4-1, the following signals can be controlled through the OPAL software:

- J input or D input
- K input or E input
- Reset input
- Output Enable



TSP-MAPL-07

Figure 4-1 Input/Output Macrocell

The equations to implement the desired functions are defined as follows:

```
Begin Header
   Title    MAPL I/O control signal
   Pattern  control
   Revision A
   Author   Tarif Arabi
   Date     10/15/90
   End Header

Begin Definition
   Inputs  In1, In2, In3, In4, In5;
   feedbacks (JK)  out1;
   outputs (DE)    out2;
End Definition

Begin Equations

         /out1.j = In1 * /In3      { J,K input control  }
                   + In4;
         out1.K  = In2;
         out1.re = In1 * In5;       { Reset control      }
         out1.oe = /In5;            { Tristate control   }

         out2.d = /In2 * In4 * out1; { D,E input control  }
         out2.re = In1* In5;         { Reset control      }
         out2.oe = In5;              { Tristate control   }

End Equations
```

Note that out1.j was defined in the above example as an active low signal. After compiling the entry file, the active low applying DeMorgan's Law will be:

```
          /out1.j = In1 * /In3 + In4
   will be equivalent to:
          out1.j =  /In1 * /In4 + In3 * /In4
```

Running "OPL2PLA CONTROL" will take CONTROL.OPL as the entry file and generate a PLA file called control.PLA :

```
   #$ TOOL NSC opl2pla B.04
   #$ TITLE         MAPL control signal
   #$ TITLE
   #$ PINS 7 In1 In2 In3 In4 In5 out1 out2
   #$ NODES 0
```

```
.i 6
.o 8
.type f
.phase 11111111
.ilb In1 In2 In3 In4 In5 out1
.ob  out2.d out2.ce out1.j out1.k out1.re out1.oe out2.re
out2.oe
0--0--   --1-----
--10--   --1-----
1---1-   ----1---
----0-   -----1--
-1----   ---1----
-0-1-1   1-------
1---1-   ------1-
----1-   -------1
--0---   -1------
.e
```

Running "FITMAPL CONTROL" will then take CONTROL.PLA as its entry file and fit the design into the MAPL128 by assigning pins and pages. The output file will be CONTROL.OUT:

```
#$ TOOL NSC
#$ TITLE         MAPL control signal
#$ TITLE
#$ DEVICE MAPL128
.i 9
.o 11
#$ PINS  7 In1:6 In2:5 In3:4 In4:3 In5:2 out1:10 out2:11
#$ NODES 3 pb2:39 pb1:38 pb0:37
.ilb In1 In2 In3 In4 In5 out1 pb2 pb1 pb0
.ob  out2.d out2.ce out1.j out1.k out1.re out1.oe out2.re
out2.oe pb2.reg pb1.reg pb0.reg
.type f
.phase 11111111111
.p 9
0--0--000 --1-----000
--10--000 --1-----000
--0---000 -1------000
-0-1-1000 1-------000
-1----000 ---1----000
----1---- -------1~~~
1---1---- ------1-~~~
----0---- -----1--~~~
1---1---- ----1---~~~
.e
```

Running "PLA2EQN CONTROL.OUT" takes CONTROL.OUT as its input file and generates an equation file CONTROL.EQN :

```
          MAPL control signal

;Translated from NSC formatted PLA file.

CHIP  filename MAPL128

CLK In5 In4 In3 In2 In1 NC NC NC out2 out1
NC NC GND NC NC NC NC NC NC NC NC NC
NC NC NC VCC
NC NC NC NC NC NC NC NC
pb0 pb1 pb2

EQUATIONS

out2 := /In2 * In4 * out1 * /pb2 * /pb1 * /pb0
out2.CE = /In3 * /pb2 * /pb1 * /pb0
out1.J = /In1 * /In4 * /pb2 * /pb1 * /pb0
      + In3 * /In4 * /pb2 * /pb1 * /pb0
out1.K = In2 * /pb2 * /pb1 * /pb0
out1.RE = In1 * In5
out1.OE = /In5
out2.RE = In1 * In5
out2.OE = In5
pb2 := GND
pb1 := GND
pb0 := GND
```

Finally running "EQN2JED CONTROL" will take CONTROL.EQN as its input file and generate a Jedec map for the MAPL128 called CONTROL.JED :

```
MAPL128
EQN2JED -- Boolean Logic to JEDEC file assembler (Version 1.10)
Copyright (R) National Semiconductor Corporation 1990
Assembled from "control.eqn". Date: 10-15-90
          MAPL control signal
*
QF14833*QP28*F0*
L0000
11101101111111111101111111111111111111111111111111111111
11111111111111111111111111111111111111111011111111111
11111011111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111110111111111
10111110111111111111111111111111111111111111111111111111
```

```
1111111111111111111111111111111111111111111110111111111
1111011011111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111110111111111
1101111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111011111111*
L14336
0111111101111111111111111111111111111111111111111111111111*
L14394
1111111110111111111111111111111111111111111111111111111111*
L14452
1111111101111111111111111111111111111111111111111111111111*
L14510
0000000000000000000000000000000000000000000000000000000000*
L14568
11111111111111111111111111*
L14595
00011111111111111111111111*
L14622
00011111111111111111111011111*
L14649
11111111111111111111111*
L14673
1111111111001111*
L14689
1111111111101111*
L14705
0000000000000000000000000000000000000000000000000000000000*
L14769
0000000000000000000000000000000000000000000000000000000000*
C67DF*
?0000
```

4.2 A 4-Bit Counter with Reset

Using OPAL, a MAPL128 can be implemented as a 4-bit counter with reset in three different ways. This application demonstrates different ways to enter the same design.

Figure 4-2 is the entry file for 4-bit counter with reset implemented in a MAPl128 using OPAL truth table entry only.

Figure 4-3 is the same design implemented in a MAPL128 using OPAL Boolean equation entry only.

Figure 4-4 is the same design implemented in a MAPL128 using OPAL state diagram entry only.

Note that any of the above entry files will give the same results. To get the JEDEC file run OPL2PLA, FITMAPL, PLA2EQN, and EQN2-JED.

```
Begin Header
        Title      4 bit counter
        Pattern    4count
        Revision   A
        Author     Tarif Arabi
        Date       10/15/90
        Everything in the header block is copied directly into
        all the files that generated by OPAL as a comment field
End Header

BEGIN DEFINITION
{ Any thing surrounded by curly brackets is considered to be a
        comment}

Device MAP1128;

INPUTS
    RESET;

feedbacks (JK,HOLD)                {Define 4 I/O as a JK}
                                   {default to hold}
    CNT4,CNT3,CNT2,CNT1;
END DEFINITION

BEGIN truth_TABLE
ttin    RESET,CNT4,CNT3,CNT2,CNT1;
ttout   CNT4,CNT3,CNT2,CNT1;

1 ---- 0000
0 ---- ---!
0 ---1 --!-
0 --11 -!--
0 -111 !---

END truth_TABLE
```

Figure 4-2 OPAL Truth Table

```
begin header
   4 bit counter design using state diagram entry format only
end header

begin definition
inputs
   reset;
statebits
   cnt4,cnt3,cnt2,cnt1;
end definition

begin state_diagram

state zero : if reset !$ 0 then one else zero;
state one : if reset !$ 0 then two else zero;
state two : if reset 1$ 0 then three else zero;
state three : if reset !$ 0 then four else zero;
state four : if reset !$ 0 then five else zero;
state five : if reset !$ 0 then six else zero;
state six : if reset !$ 0 then seven else zero;
state eight : if reset !$ 0 then nine else zero;
state nine : if reset !$ 0 then ten else zero;
state ten : if reset !$ 0 then eleven else zero;
state eleven : if reset !$ 0 then twelve else zero;
state twelve : if reset !$ 0 then thirteen else zero;
state thirteen : if reset !$ 0 then fourteen else zero;
state fourteen : if reset !$ 0 then fifteen else zero;
state fifteen : goto zero;

end state_diagram
```

Figure 4-3 State Diagram Entry Format

```
begin header
   4 bit counter design using boolean equations only
end header

begin Definition

inputs reset;
feedbacks    cnt4,cnt3,cnt2,cnt1;

end Definition

begin Equations

cnt4   := /reset * /cnt4 * cnt3 * cnt2 * cntl
        + /reset * cnt4 * /cnt1
        + /reset * cnt4 * /cnt3
        _ /reset * cnt4 * /cnt2;
cnt3   := /reset * /cnt3 * cnt2 * cnt1
        _ /reset * cnt3 * /cnt2
        + /reset * cnt3 * /cnt1;
cnt2   := /reset * /cnt2 * cnt1
        + /reset * cnt2 * /cnt1;
cnt1   := /reset * /cnt1;

end Equations
```

Figure 4-4 Boolean Equations

4.3 A 3-bit Up-Down Counter With 7-segment Display Output.

This application demonstrates the use of OPAL's state diagram entry format and set assignment to implement this design. The counter will reset to zero if rst is low, It will count:

- upward if up is high,
- downward if down is high,
- upward if both are high,
- hold value if both are low.

This application shows how to implement the above mentioned design using state diagram entry. This design can also be entered using the truth table format.

3_to_7.OPL

```
begin header
      Company     :  National Semiconductor Corp.
      Author      :  Moutaz Kotob
      Date        :  08/22/90
      Revision    :  A
      Description:  A three-bit up-down counter with 7-segment
                    display output.

end header

begin definition
      device mapl128;
      inputs up, down, rst;

   { 7-segment display output}
      outputs  a, b, c, d, e, f, g;
```

```
   { symbols to define state names and their values, statebits
     will be assigned automatically}
      symbols zero=0, one=^b1, two=^b10, three=3, four=4,
         five=5, six=6, seven=7;

   { Sets are defined to group inputs and outputs}
      Sets out_disp= [a,b,c,d,e,f,g], ZERO= [1,1,1,1,1,1,0],
      TWO  = [1,1,0,1,1,0,1], THREE= [1,1,1,1,0,0,1],
      FOUR = [0,1,1,0,0,1,1], FIVE = [1,0,1,1,0,1,1],
      SIX  = [1,0,1,1,1,1,1], SEVEN= [1,1,1,0,0,0,0];
end definition

begin state_diagram

        state zero :
           out_disp = ZERO;      { out '0' to 7-segment display}
           if /rst then zero     { reset counter }
           else if up then one   { count upward }
           else if down then seven  { count downward }
           else zero;            { hold }

        state one :
           out_disp = [0,1,1,0,0,0,0];  { set notation was used
                                          rather than a defined
                                        set  to display '1'}
           if /rst then zero
           else if up then two
           else if down then zero
           else one;

        state two :
           out_disp = TWO;
           if /rst then zero
           else if up then three
           else if down then one
           else two;

        state three :
           out_disp = THREE;
           if /rst then zero
           else if up then four
           else if down then two
           else three;
```

```
        state four :
           out_disp = FOUR;
           if /rst then zero
           else if up then five
           else if down then three
           else four;

        state five :
           out_disp = FIVE;
           if /rst then zero
           else if up then six
           else if down then four
           else five;

        state six :
           out_disp = SIX;
           if /rst then zero
           else if up then seven
           else if down then five
           else six;

        state seven :
           out_disp = SEVEN;
           if /rst then zero
           else if up then zero
           else if down then six
           else seven;

end state_diagram
```

3_to_7.PLA

```
#$ TOOL NSC nsc2pla B.04
#$ TITLE    Company    :  National Semiconductor Corp.
#$ TITLE    Author     :  Moutaz Kotob
#$ TITLE    Date       :  08/22/90
#$ TITLE    Revision   :  A
#$ TITLE       Description:  A three-bit up-down counter with
7-segment display
#$ TITLE  output.
#$ TITLE
#$ TITLE
#$ DEVICE mapl128
#$ PINS 10 up down rst a b c d e f g
#$ NODES 3 statebit03 statebit02 statebit01
.i 6
.o 10
```

```
.type f
.phase 1111111111
.ilb up down rst statebit03 statebit02 statebit01
.ob  statebit03.reg statebit02.reg statebit01.reg a.reg b.reg
c.reg d.reg
 e.reg f.reg g.reg
--- 000 --- 1111110
--0 000 000 -------
1-1 000 001 -------
011 000 111 -------
001 000 000 -------
--- 001 --- 0110000
--0 001 000 -------
1-1 001 010 -------
011 001 000 -------
001 001 001 -------
--- 010 --- 1101101
--0 010 000 -------
1-1 010 011 -------
011 010 001 -------
001 010 010 -------
--- 011 --- 1111001
--0 011 000 -------
1-1 011 100 -------
011 011 010 -------
001 011 011 -------
--- 100 --- 0110011
--0 100 000 -------
1-1 100 101 -------
011 100 011 -------
001 100 100 -------
--- 101 --- 1011011
--0 101 000 -------
1-1 101 110 -------
011 101 100 -------
001 101 101 -------
--- 110 --- 1011111
--0 110 000 -------
1-1 110 111 -------
011 110 101 -------
001 110 110 -------
--- 111 --- 1110000
--0 111 000 -------
1-1 111 000 -------
011 111 110 -------
001 111 111 -------
.e
```

4.4 15-Bit Up/down Counter

This application demonstrates the use of the MAPL128 in a state machine design to implement a 15 bit up/down counter with reset and hold. The JEDEC map for the MAPL128 was generated using OPAL software. OPAL includes FITMAPL.EXE that accomplishes automatic paging of the entry file. However, in cases where there is a large number of "don't cares" in the feedback terms, FITMAPL may not be able to find a solution. This example implements a 15bit up/down counter and is a case where FITMAPL doesn't find a solution.

This manual paging example shows how a designer can use OPAL to generate JEDEC maps for the MAPL products by completely specifying all pins and page data thereby eliminating the need to execute FITMAPL.

The design was simulated using the Verilog simulator. This application shows the implementation of the design in a truth table, while if the same design was entered using the state diagram entry format, 32,768 states would appear in the entry file.

To implement the same design in a regular PAL or GAL device would take roughly four PAL16R4's. While in this application, 25% of the MAPL128 was used to implement the 15-bit counter (2 pages out of 8).

To get the JEDEC file run the following modules:

- OPL2PLA 15UPDN
- PLA2EQN 15UPDN
- EQN2JED 15UPDN

```
{
 Entry file for a manual paging example 15bit up down counter.
}
Begin Header
   Title     15 bit up/down counter
   Pattern   15UPDN
   Revision  D
   Author    Tarif Arabi
   Date      12/10/90

   Everything in the header command is copied directly into the
   jedec map as a comment field
End Header

BEGIN DEFINITION
{
Any thing surrounded by curly brackets is considered to be a com-
ment
}

Device MAPL128;                    { Specify the device used    }

INPUTS                             { Define 3 inputs            }

       RESET=6,  HOLD=5, UP_DOWN=4;

feedbacks (JK, HOLD, buried)  {Define 3 buried register as count}
                          {bits configured as JK with Hold default}

       BCNT14=29,BCNT13=30,BCNT12=31;

feedbacks (JK,HOLD)                {Define 11 I/O as a JK       }
                                   {default to hold             }


CNT11=11,CNT10=10,CNT9=9,CNT8=8,CNT7=7,CNT6=17,CNT5=18,CNT4=19,
        CNT3=20,CNT2=21,CNT1=22;

feedbacks (JK,TOGGLE)              {Define one I/O as JK toggling}
                                   {always with the clock       }
CNT0=23;

OUTPUTS (JK, HOLD)     {Define 3 outputs as JK so the total number
}
                    {of outputs will be 15.  Observe that in the  }
                    {truth table, they are driven by the buried   }
                   {feedback, which actually represents the count}
```

```
        CNT14=12,CNT13=13,CNT12=15;

feedbacks (JK, HOLD, buried)  {Define the 3 page bit labels }
                              {(8 pages) by using these node numbers}
        P2=39,P1=38,P0=37;

END DEFINITION
{
For this example we will use truth table entry to map the pages
manualy and fit the design in the MALP128
}

BEGIN truth_TABLE

{ Truth table inputs, outputs  (ttin, ttout) }

ttin    RESET,  HOLD, UP_DOWN, BCNT14,BCNT13,BCNT12,

CNT11,CNT10,CNT9,CNT8,CNT7,CNT6,CNT5,CNT4,CNT3,CNT2,CNT1,CNT0,
        P2,P1,P0;

ttout   BCNT14,BCNT13,BCNT12,

CNT11,CNT10,CNT9,CNT8,CNT7,CNT6,CNT5,CNT4,CNT3,CNT2,CNT1,CNT0,
        P2,P1,P0,CNT14,CNT13,CNT12;

{ Page Zero count up }
01- ---------------- 000 ???????????????? 000  ??? { ? = Hold       }
000 ---------------1 000 --------------!- 000  --- { ! = Toggle     }
000 --------------11 000 -------------!-- 000  --- {- = Don't care}
000 -------------111 000 ------------!--- 000  ---
000 ------------1111 000 -----------!---- 000  ---
000 -----------11111 000 ----------!----- 000  ---
000 ----------111111 000 ---------!------ 000  ---
000 ---------1111111 000 --------!------- 000  ---
000 --------11111111 000 -------!-------- 000  ---
000 -------111111111 000 ------!--------- 000  ---
000 ------1111111111 000 -----!---------- 000  ---
000 -----11111111111 000 ----!----------- 000  ---
000 ----111111111111 000 ---!------------ 000  --!
000 ---1111111111111 000 --!------------- 000  -!-
000 --11111111111111 000 -!-------------- 000  !--
000 -111111111111111 000 !--------------- 000  ---
001 ---------------- 000 ---------------? 001  ---
```

```
{ Page One count down }
01- --------------- 001 ?????????????? 001  ???
001 --------------0 001 -------------!- 001  ---
001 -------------00 001 ------------!-- 001  ---
001 ------------000 001 -----------!--- 001  ---
001 -----------0000 001 ----------!---- 001  ---
001 ----------00000 001 ---------!----- 001  ---
001 ---------000000 001 --------!------ 001  ---
001 --------0000000 001 -------!------- 001  ---
001 -------00000000 001 ------!-------- 001  ---
001 ------000000000 001 -----!--------- 001  ---
001 -----0000000000 001 ----!---------- 001  ---
001 ----00000000000 001 ---!----------- 001  ---
001 ---000000000000 001 --!------------ 001  --!
001 --0000000000000 001 -!------------- 001  -!-
001 -00000000000000 001 !-------------- 001  !--
000 --------------- 001 --------------? 000  ---

END truth_TABLE

BEGIN equations

{This block is used to control the reset function in MAPL.      }
{Note that the 16 product terms in the page were used for       }
{the counting and the reset function is implemented separatly   }

BEGIN equations

    [BCNT14,BCNT13,BCNT12,CNT11,CNT10,CNT9].RE=RESET;
    [CNT8,CNT7,CNT6,CNT5,CNT4].RE=RESET;
    [CNT3,CNT2,CNT1,CNT0,P2,P1,P0,CNT14,CNT13,CNT12].RE=RESET;

END equations
```

15UPDN.PLA

```
#$ TOOL NSC nsc2pla B.04
#$ TITLE
#$ TITLE          Title      15 bit up/down counter
#$ TITLE          Pattern    15UPDN
#$ TITLE          Revision   D
#$ TITLE          Author     Tarif Arabi
#$ TITLE          Date       12/10/90
#$ TITLE
#$ TITLE          This application is a 15 bit UP/DOWN counter
#$ TITLE          with reset and hold.
#$ TITLE
#$ TITLE          Everything in the header command is copied
#$ TITLE          directly into the jedec map as a comment field
#$ TITLE
#$ DEVICE MAPL128
#$ PINS 18 RESET:6 HOLD:5 UP_DOWN:4 CNT11:11 CNT10:10 CNT9:9
CNT8:8 CNT7:7 CNT6:
#$ NODES 6 BCNT14:29 BCNT13:30 BCNT12:31 P2:39 P1:38 P0:37
.i 21
.o 63
.type f
.phase 111111111111111111111111111111111111111111111111111111111
11111111
.i1b RESET HOLD UP_DOWN BCNT14 BCNT13 BCNT12 CNT11 CNT10 CNT9
CNT8 CNT7 CNT6 CNT5 CNT4 CNT3 CNT2 CNT1 CNT0 P2 P1 P0
.ob CNT14.j CNT14.k CNT13.j CNT13.k CNT12.j CNT12.k BCNT14.j
BCNT14.k BCNT13.j BCNT13.k BCNT12.j BCNT12.k CNT11.j CNT11.k
CNT10.j CNT10.k CNT9.j CNT8.j CN5.j CNT7.k CNT6.j CNT6.k CNT5.j
CNT5.k CNT4.j CNT4.k CNT3.j CNT3.k CNT2.j CNT2.k CNT1.j CNT1.k
CNT0.j- CNT0.k- P2.j P2.k P1.j P1.k P0.j P0.k CNT9.RE CNT10.RE
CNT11.RE BCNT12.RE BCNT13.RE BCNT14.RE CNT4.RE CNT5.RE CNT6.RE
CNT7.RE CNT8.RE CNT12.RE CNT13.RE CNT14.RE P0.RE P1.RE P2.RE CN-
T0.RE CNT1.RE CNT2.RE CNT3.RE
1-------------------     ----------------------------------------1----------
1-------------------     ----------------------------------------1----------
1-------------------     ----------------------------------------1----------
1-------------------     ----------------------------------------1----------
1-------------------     ----------------------------------------1----------
1-------------------     ----------------------------------------1----------
1-------------------     ----------------------------------------1----------
1-------------------     ----------------------------------------1----------
1-------------------     ----------------------------------------1----------
1-------------------     ----------------------------------------1----------
1-------------------     ----------------------------------------1----------
1-------------------     ----------------------------------------1----------
1-------------------     ----------------------------------------1----------
1-------------------     ----------------------------------------1----------
1-------------------     ----------------------------------------1----------
```

```
1--------------------     --------------------------------------------------------
1--------------------     --------------------------------------------------------
1--------------------     --------------------------------------------------------
1--------------------     --------------------------------------------------------
1--------------------     --------------------------------------------------------
1--------------------     --------------------------------------------------------
01----------------000     000000000000000000000000000000000011010101--------------
000--------------1000     --------------------------------11--010101--------------
000-------------11000     ------------------------------11----010101--------------
000------------111000     ----------------------------11------010101--------------
000-----------1111000     --------------------------11--------010101--------------
000----------11111000     ------------------------11----------010101--------------
000---------111111000     ----------------------11------------010101--------------
000--------1111111000     --------------------11--------------010101--------------
000-------11111111000     ------------------11----------------010101--------------
000------111111111000     ----------------11------------------010101--------------
000-----1111111111000     --------------11--------------------010101--------------
000----11111111111000     ------------11----------------------010101--------------
000---111111111111000     ----11----11----------------------010101--------------
000--1111111111111000     --11----11------------------------010101--------------
000-11111111111111000     11----11--------------------------010101--------------


001---------------000     ----------------------------------11010110--------------
01----------------001     000000000000000000000000000000000011010111--------------
001--------------0001     --------------------------------11--010101--------------
001-------------00001     ------------------------------11----010101--------------
001------------000001     ----------------------------11------010101--------------
001-----------0000001     --------------------------11--------010101--------------
001----------00000001     ------------------------11----------010101--------------
001---------000000001     ----------------------11------------010101--------------
001--------0000000001     --------------------11--------------010101--------------
001-------00000000001     ------------------11----------------010101--------------
001------000000000001     ----------------11------------------010101--------------
001-----0000000000001     --------------11--------------------010101--------------
001----00000000000001     ----11----11----------------------010101--------------
001---000000000000001     --11----11------------------------010101--------------
001--0000000000000001     11----11--------------------------011101--------------
001-00000000000000001     ----------------------------------11011101--------------
.e
```

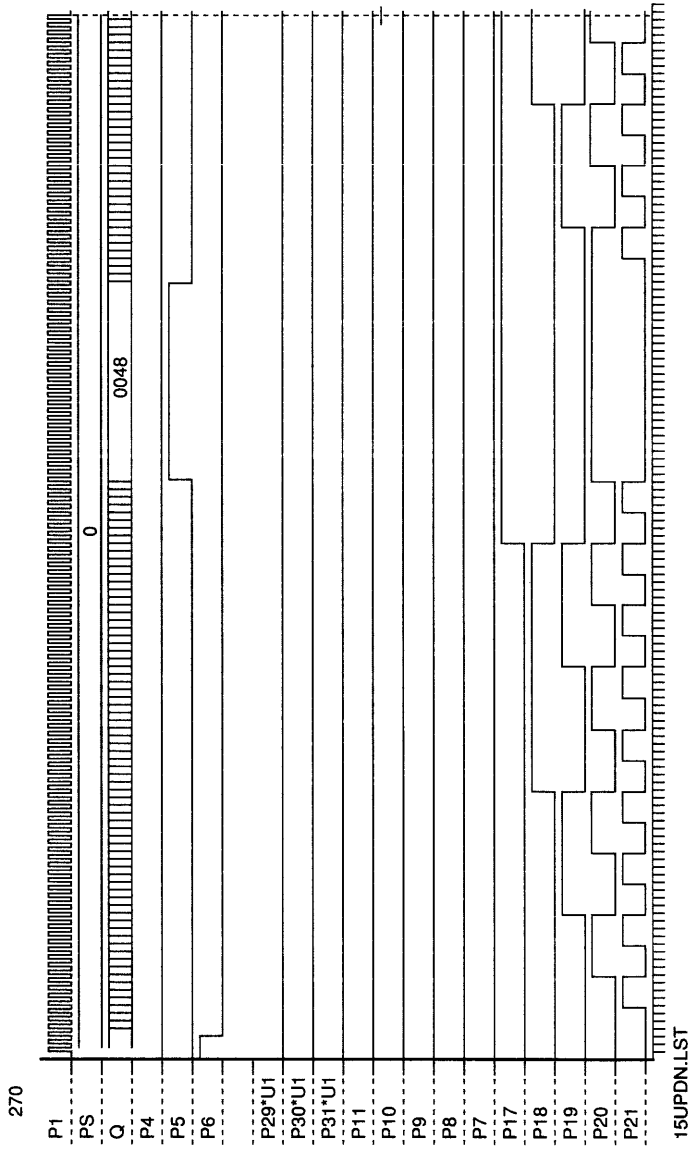
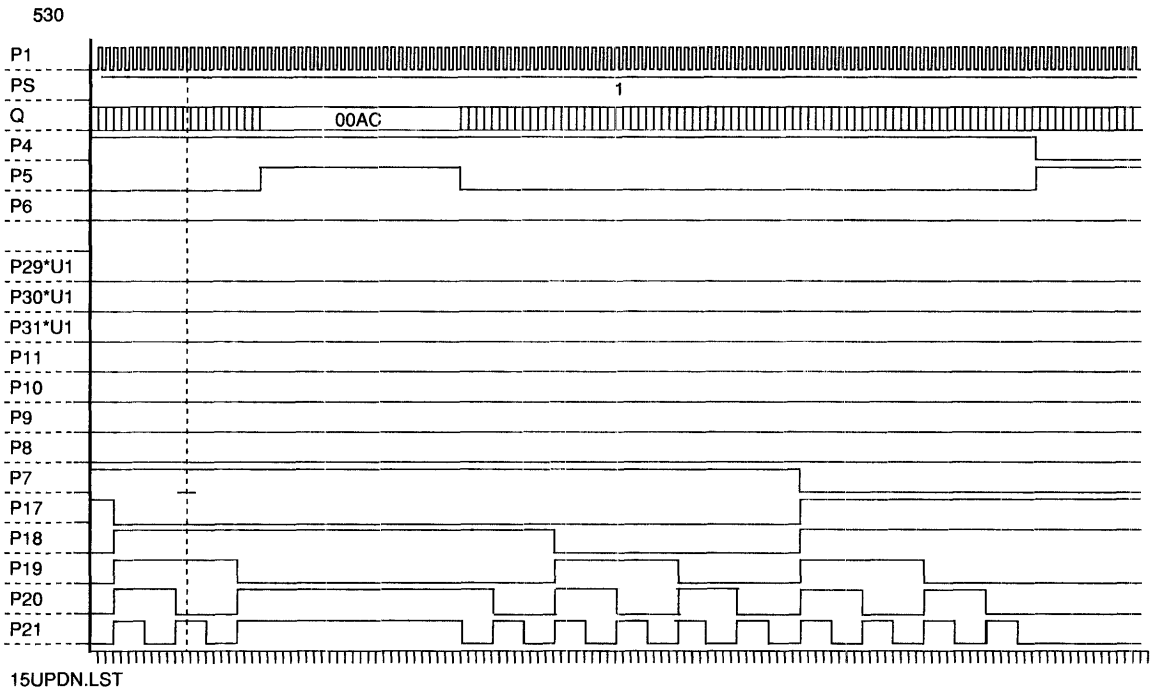
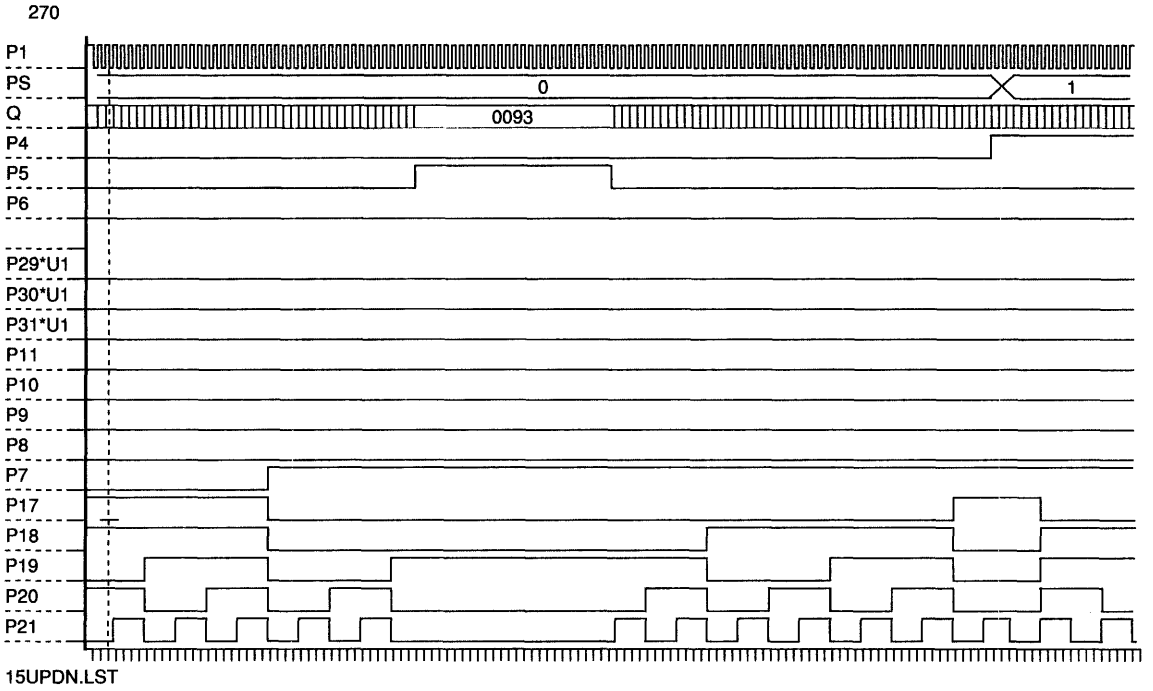


Figure 4-5 Timing Diagram



TSP-MAPL-10

Figure 4-6 Timing Diagram



TSP-MAPL-09

Figure 4-7 Timing Diagram

4.5 12-Bit L/R Shift Register With Load

This application demonstrates the use of the MAPL128 in a state machine design to implement a 12 bit shift register with reset and load. The design will shift right, shift left or load the 12-bit register. The JEDEC map for the MAPL128 was generated using OPAL software. OPAL includes FITMAPL.EXE which performs automatic paging on the entry file. However, in cases where there is a large number of "don't cares" in the feedback terms, FITMAPL may not be able to find a solution. This example implements a "12-bit" shift left/right register and is a case where FITMAPL doesn't find a solution.

This example shows how a designer can use OPAL to generate JEDEC maps for the MAPL products by completely specifying only the page data and using FITMAPL to generate the PIN list The design was simulated using the Verilog simulator. In this example 40% of the MAPL128 was used to implement the "12-bit" shift register (3 pages out of 8).

To get the JEDEC file run the following modules:

- OPL2PLA 12SHIFT
- FITMAPL 12SHIFT
- PLA2EQN 12SHIFT
- EQN2JED 12SHIFT

```
{
 Entry file for a 12bit shift register - manual paging example.
}
Begin Header

Title    12 bits shift right/left with load
Pattern  12shift
Revision C
Author   Tarif Arabi
Date     12/10/90

Everything in the header command is copied directly into the
jedec map as a comment field

End Header
```

```
BEGIN DEFINITION
{
Any thing surrounded by curly brackets are considered to be a
comment
 }

Device MAPL128;

INPUTS                    { Define 5 inputs              }

    RESET,  HOLD, SH, LD, SHIN;

FEEDBACKS (DE,RST)        {Define 12 I/O as DE register }
                          {default to reset              }

        S11,S10,S9,S8,S7,S6,S5,S4,S3,S2,S1,S0;

FEEDBACKS (JK, HOLD, BURIED)

    {Define the page labels because manual paging is used}

       p2=39,p1=38,p0=37;

END DEFINITION


BEGIN TRUTH_TABLE

ttin RESET,  HOLD,SH,LD,SHIN,S11,S10,S9,S8,S7,
             S6,S5,S4,S3,S2,S1,S0,p2,p1,p0;

ttout  S11,S10,S9,S8,S7,S6,S5,S4,S3,S2,S1,S0,p2,p1,p0;
```

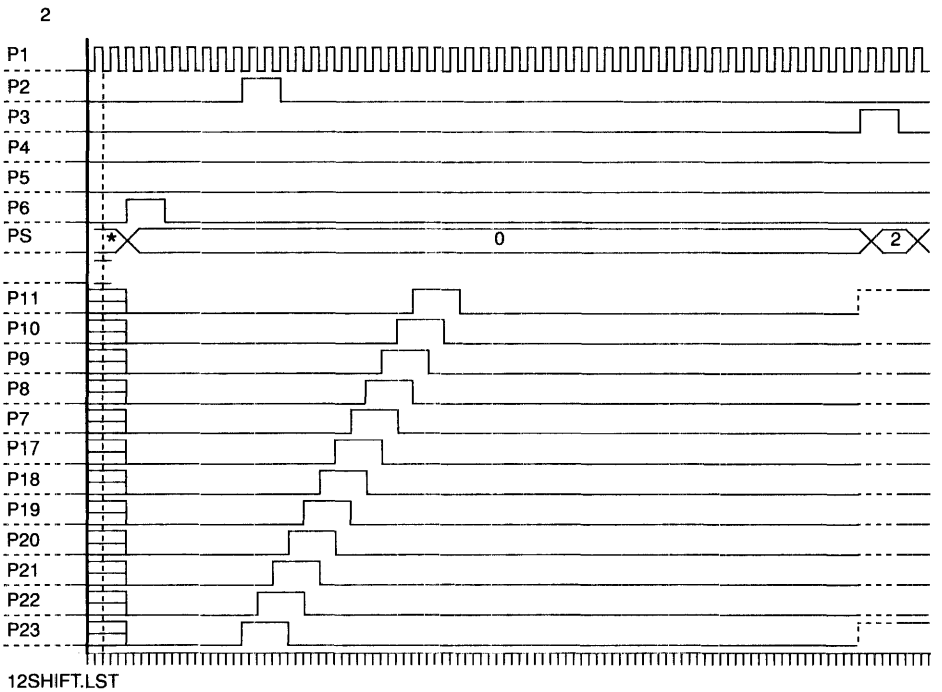
```
{
Input  I/O         Page   Next I/O  Next
                                    page  }
01000 ------------ 000 ???????????? 000
00001 ------------ 000 -----------1 000   {Shift left when sh = L}
0000- -----------1 000 ----------1- 000
0000- ----------1- 000 ---------1-- 000
0000- ---------1-- 000 --------1--- 000
0000- --------1--- 000 -------1---- 000
0000- -------1---- 000 ------1----- 000
0000- ------1----- 000 -----1------ 000
0000- -----1------ 000 ----1------- 000
0000- ----1------- 000 ---1-------- 000
0000- ---1-------- 000 --1--------- 000
0000- --1--------- 000 -1---------- 000
0000- -1---------- 000 1----------- 000
0010- ------------ 000 ???????????? 001
0001- ------------ 000 ------------ 010

01--- ------------ 001 ???????????? 001
00101 ------------ 001 1----------- 001 {Shift right when SH = H}
0010- ----------1- 001 -----------1 001
0010- ---------1-- 001 ----------1- 001
0010- --------1--- 001 ---------1-- 001
0010- -------1---- 001 --------1--- 001
0010- ------1----- 001 -------1---- 001
0010- -----1------ 001 ------1----- 001
0010- ----1------- 001 -----1------ 001
0010- ---1-------- 001 ----1------- 001
0010- --1--------- 001 ---1-------- 001
0010- -1---------- 001 --1--------- 001
0010- 1----------- 001 -1---------- 001
0000- ------------ 001 ???????????? 000
0001- ------------ 001 ------------ 010
```

```
01--- ------------ 010 ???????????? 010
0001- -----------1 010 -----------1 010  {Load when LD = H  }
0001- ----------1- 010 ----------1- 010
0001- ---------1-- 010 ---------1-- 010
0001- --------1--- 010 --------1--- 010
0001- -------1---- 010 -------1---- 010
0001- ------1----- 010 ------1----- 010
0001- -----1------ 010 -----1------ 010
0001- ----1------- 010 ----1------- 010
0001- ---1-------- 010 ---1-------- 010
0001- --1--------- 010 --1--------- 010
0001- -1---------- 010 -1---------- 010
0001- 1----------- 010 1----------- 010
0000- ------------ 010 ???????????? 000
0010- ------------ 010 ???????????? 001

END TRUTH_TABLE

   {Disable the I/O when loading (LD = 1)  }
   {I/O used as input to load the new value}

BEGIN equation

 [S11,S10,S9,S8,S7,S6,S5,S4,S3,S2,S1,S0].oe= /LD;

END equation
```

12SHIFT.PLA

```
#$ TOOL NSC nsc2pla B.04
#$ TITLE
#$ TITLE          Title     12 bits shift right/left with load
#$ TITLE          Pattern   12shift
#$ TITLE          Revision  C
#$ TITLE          Author    Tarif Arabi
#$ TITLE          Date      12/10/90
#$ TITLE
#$ TITLE          This application is a 12 bit shift right/left
#$ TITLE          and load with reset and hold.
#$ TITLE
#$ TITLE          Everything in the header command is copied
#$ TITLE          directly into the jedec map as a comment field
#$ TITLE
#$ DEVICE MAPL128
#$ PINS 17 RESET HOLD SH LD SHIN S11 S10 S9 S8 S7 S6 S5 S4 S3 S2
S1 S0
.i 20
.o 42
.type f
.phase 11111111111111111111111111111111111111111111

---0----------------      ------------------------------1-----------
---0----------------      -------------------------------1----------
---0----------------      --------------------------------1---------
---0----------------      ---------------------------------1--------
---0----------------      ----------------------------------1-------
---0----------------      -----------------------------------1------
---0----------------      ------------------------------------1-----
---0----------------      -------------------------------------1----
---0----------------      --------------------------------------1---
---0----------------      ---------------------------------------1--
---0----------------      ----------------------------------------1-
---0----------------      -----------------------------------------1
01000------------000      -1-1-1-1-1-1-1-1-1-1-1-1010101------------
00001------------000      ----------------------10010101------------
0000------------1000      --------------------10--010101------------
0000-----------1-000      ------------------10----010101------------
0000----------1--000      ----------------10------010101------------
0000---------1---000      --------------10--------010101------------
0000--------1----000      ------------10----------010101------------
0000-------1-----000      ----------10------------010101------------
0000------1------000      --------10--------------010101------------
0000-----1-------000      ------10----------------010101------------
0000----1--------000      ----10------------------010101------------
0000---1---------000      --10--------------------010101------------
0000--1----------000      10----------------------010101------------
0010-------------000      -1-1-1-1-1-1-1-1-1-1-1-1010111------------
0001-------------000      ------------------------011001------------
```

```
01---------------01       -1-1-1-1-1-1-1-1-1-1-1-1010110-----------
00101------------001       10---------------------010110-----------
0010-----------1-001       ----------------------10010110-----------
0010----------1--001       --------------------10--010110-----------
0010---------1---001       ------------------10----010110-----------
0010--------1----001       ----------------10------010110-----------
0010-------1-----001       --------------10--------010110-----------
0010------1------001       ------------10----------010110-----------
0010-----1-------001       ----------10------------010110-----------
0010----1--------001       --------10--------------010110-----------
0010---1---------001       ------10----------------010110-----------
0010--1----------001       ----10------------------010110-----------
0010-1-----------001       --10--------------------010110-----------
0000-------------001       -1-1-1-1-1-1-1-1-1-1-1-1010101-----------
0001-------------001       ------------------------011001-----------
01---------------010       -1-1-1-1-1-1-1-1-1-1-1-1011001-----------
0001------------1010       ----------------------10011001-----------
0001-----------1-010       --------------------10--011001-----------
0001----------1--010       ------------------10----011001-----------
0001---------1---010       ----------------10------011001-----------
0001--------1----010       --------------10--------011001-----------
0001-------1-----010       ------------10----------011001-----------
0001------1------010       ----------10------------011001-----------
0001-----1-------010       --------10--------------011001-----------
0001----1--------010       ------10----------------011001-----------
0001---1---------010       ----10------------------011001-----------
0001--1----------010       --10--------------------011001-----------
0001-1-----------010       10---------------------011001-----------
0000-------------010       -1-1-1-1-1-1-1-1-1-1-1-1010101-----------
0000-------------010       -1-1-1-1-1-1-1-1-1-1-1-1010110-----------
.e
```



TSP-MAPL-11

Figure 4-8 Timing Diagram

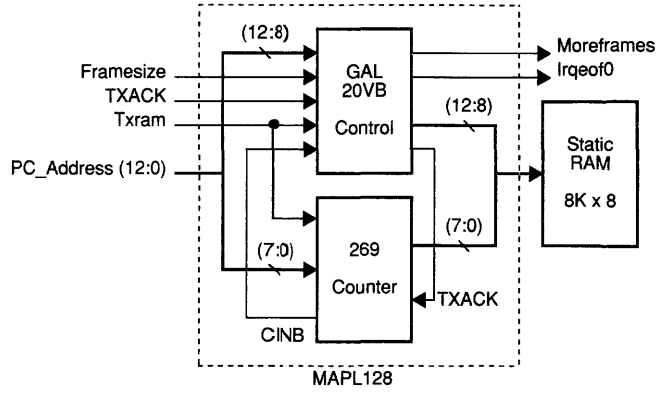
4.6 High Speed Frame Buffer Control

Introduction

This application demonstrates how the MAPL family can integrate PAL, GAL, and discrete logic devices. Specifically, this application note describes the merging of a GAL20V8 and a counter into a MAPL128 on the FDDI MAC Layer Evaluation Board with a reduction in power and board space. The integration of functions is done using OPAL software along with a multiple level logic partitioning algorithm. It is assumed that the reader is already familiar with the FDDI MAC Layer Evaluation Board and multiple level logic partitioning.

Description Of Design

The block diagram in Figure 4-9 shows which devices and I/O signals are merged into one MAPL128 device. The GAL portion of the initial design contains the upper five bits of the address which accesses the SRAM, along with some extra decode functions. The input control signals are received from the mode register (FRAMESIZE), the PC interface module (TXRAM), and the BMAC (TXACK). The design outputs a IRQEOF0 signal back to the BMAC, and drives the MOREFRAMES signal for the transmit se-



TSP-MAPL-12

Figure 4-9 Block Diagram

quencer. Figure 4-10 shows where this block fits into the entire block diagram of the FDDI MAC Layer Evaluation Board.

The MAPL design also cuts down on a few I/O signals between the GAL and the counter. The three I/O signals that are integrated within the MAPL design are CINB, TXACK~, and TXRAM~. The CINB signal is the carry in bit from the 269 which triggers the GAL to count through the high order address bits. The signal TXACK is inverted through the GAL and then fed to the counter. TXRAM~ is used as input to both blocks and thus can be integrated. So, not only can the MAPL reduce board space and power consumption, it can also eliminate some PCB routing.

Figure 4-11 shows the implementation of the design in OPAL's high end .OPL file format. The .OPL file gives the designer freedom to choose state machine language, multilevel boolean equations, enhanced truth table functions, or any combination of the three to implement a design. For this specific design, it is easier to

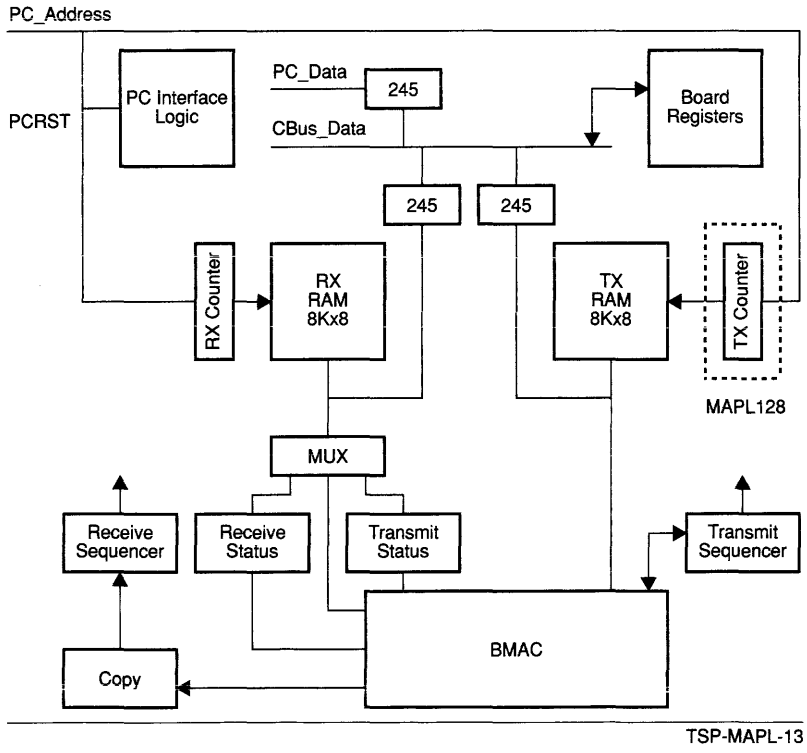


Figure 4-10 FDDI MAC Board Block Diagram Data Path

```
begin header
      transmit counter
end header

begin definition

      inputs
      txram,txack,framesize,pcin;
      feedbacks (JK,HOLD)
      pc12,pc11,pc10,pc9,pc8,pc7,pc6,pc5,pc4,pc3,pc2,pc1,
      cinb;
      feedbacks (JK,TOGGLE)
      p0;
      feedbacks (JK,RST)
      more;
      outputs (JK,RST)
      irqeof0,moreframes;
      outputs (JK,TOGGLE)
      pc0;

end definition

begin equation


[pc0,pc1,pc2,pc3,pc4,pc5,pc6,pc7,pc8,pc9,pc10,pc11,pc12].oe =
txram;

end equation

begin truth_table

      ttin
txram,txack,framesize,pc12,pc11,pc10,pc9,pc8,pc7,pc6,pc5,pc4,
pc3,pc2,pc1,p0,pcin,more,cinb;
      ttout
pc12,pc11,pc10,pc9,pc8,pc7,pc6,pc5,pc4,pc3,pc2,pc1,p0,
pc0,more,moreframes,irqeof0,cinb;
```

Figure 4-11 Transmit Counter
(Sheet 1 of 2)

```
0-- -------------1-0 ------------11---0
0-- -----------1---0 -----------1-----0
0-- ----------1----0 ----------1------0
0-- ---------1-----0 ---------1-------0
0-- --------1------0 --------1--------0
0-- -------1-------0 -------1---------0
0-- ------1--------0 ------1----------0
0-- -----1---------0 -----1-----------0
0-- ----1----------0 ----1------------0
0-- ---1-----------0 ---1-------------0
0-- --1------------0 --1--------------0
0-- -1-------------0 -1---------------0
0-- 1--------------0 1----------------0
0-- ---------------0 -----------------0
11- ------------0--0 -----------!--1110
11- -----------00--0 ----------!---1110
11- ----------000--0 ---------!----1110
11- ---------0000--0 --------!-----1110
11- --------00000--0 -------!------1110
11- -------000000--0 ------!-------1110
11- ------0000000--0 -----!-----------0
11- -----00000000--0 ----!-----------11
11- ----000000000--0 ---!------------11
11- ---0000000000--0 --!-------------11
11- --00000000000--0 -!--------------11
11- -000000000000--0 !---------------11
1-- ----1---------11 -------------11-0
1-- ---1----------11 -------------11-0
1-- --1-----------11 -------------11-0
1-- -1------------11 -------------11-0
1-- 1-------------11 -------------11-0
11- --------------01 -------------11-0
-1- --------------10 -------------11-0
--1 1--------------1 ----------------10
--1 -1-------------1 ----------------10
--1 --1------------1 ----------------10
--1 ---1-----------1 ----------------10
--- ----1----------1 ----------------10
--- ---------------0 ----------------10
-0- ---------------0 ----------------10

end truth_table
```

Figure 4-11 Transmit Counter
(Sheet 2 of 2)

enter a truth table function for a 13-bit counter instead of a numerous amount of Boolean equations. Notice that in the .OPL file for this design, Boolean equations are used along with the truth table function for output enables. Actually, two functions were integrated into the .OPL file; namely the GAL control logic and the counter.

The file in Figure 4-11, will not fit into the MAPL device due to partitioning problems. Refer to "Integrating Multiple Functions into a High Density PLD Using Efficient Mapping Algorithms" in the software section. Thus, the multiple level logic partitioning algorithm must be performed to allow the PLA file to be partitioned. An XLT term is added to the .OPL file which is shown in Figure 4-12. Once the multiple level logic partitioning algorithm is performed, the MAPL fitter program will partition the logic and allocate it to "pages" in the device. The output from the fitter program is illustrated in Figure 4-13. The only difference between this file and a PLA file is that the fitter program appends page bits to the input and output columns to allow allocation of logic and transitions from page to page within the device.

The output from the fitter program can then be used with the OPAL software to obtain an equations file as well as a JEDEC map. Using the JEDEC map and the OPALSIM simulator package, the design can be tested and verified. The OPALSIM output waveforms are illustrated in Figure 4-14 through Figure 4-16.

```
begin header
   transmit counter
end header

begin definition

   inputs
      txram,txack,framesize,pcin;
   feedbacks (JK,HOLD)
      pc12,pc11,pc10,pc9,pc8,pc7,pc6,pc5,pc4,pc3,pc2,pc1,
      cinb;
   feedbacks (JK, RST)
      pg;
   feedbacks (JK,TOGGLE)
      p0;
   feedbacks (JK,RST)
      more;
   outputs (JK,RST)
      irqeof0,moreframes;
   outputs (JK,TOGGLE)
      pc0;

end definition

begin equation

     [pc0,pc1,pc2,pc3,pc4,pc5,pc6,pc7,pc8,pc9,pc10,pc11,
pc12].oe = txram;

end equation

begin truth_table

   ttin txram,txack,framesize,pc12,pc11,pc10,pc9,pc8,pc7,pc6,
      pc5,pc4,pc3,pc2,pc1,p0,pcin,more,pg,cinb;
   ttout pc12,pc11,pc10,pc9,pc8,pc7,pc6,pc5,pc4,pc3,
      pc2,pc1,p0,pc0,more,moreframes,irqeof0,pg,cinb;
```

Figure 4-12 Transmit Counter XLT
(Sheet 1 of 2)

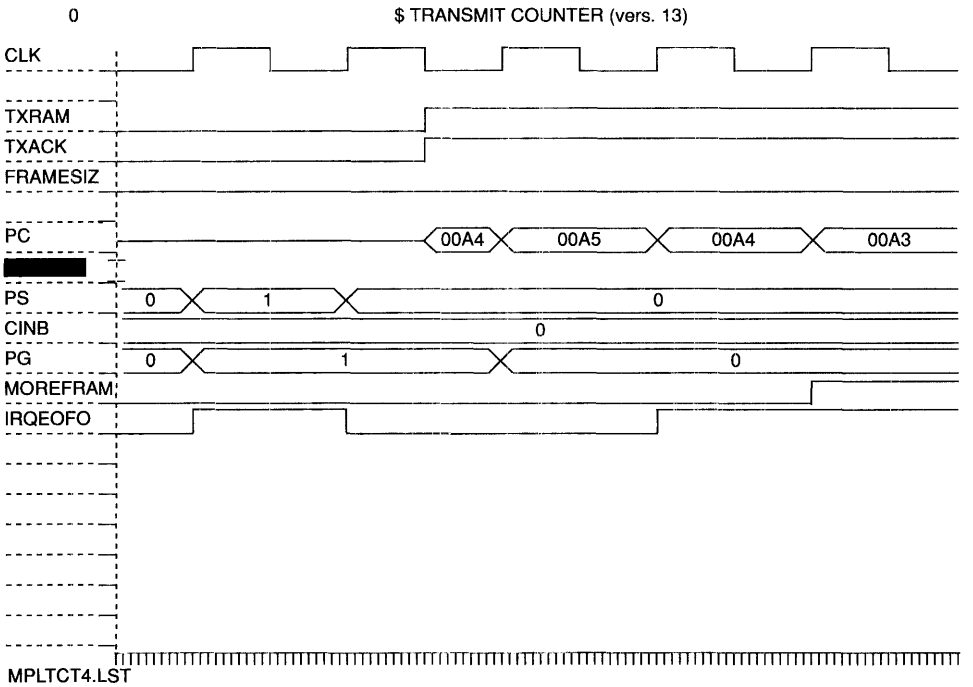
```
   0-- ------------1-10 ------------11---10
   0-- -----------1---10 -----------1-----10
   0-- ----------1----10 ----------1------10
   0-- ---------1-----10 ---------1-------10
   0-- --------1------10 --------1--------10
   0-- -------1-------10 -------1---------10
   0-- ------1--------10 ------1----------10
   0-- -----1---------10 -----1-----------10
   0-- ----1----------10 ----1------------10
   0-- ---1-----------10 ---1-------------10
   0-- --1------------10 --1--------------10
   0-- -1-------------10 -1---------------10
   0-- 1--------------10 1----------------10
   0-- ---------------00 -----------------10
   11- ------------0--00 -----------!--11100
   11- -----------00--00 ----------!---11100
   11- ----------000--00 ---------!----11100
   11- ---------0000--00 --------!-----11100
   11- --------00000--00 -------!------11100
   11- -------000000--00 ------!-------11100
   11- ------0000000--00 -----!-----------00
   11- -----00000000--00 ----!-----------101
   11- ----000000000--00 ---!------------101
   11- ---0000000000--00 --!-------------101
   11- --00000000000--00 -!--------------101
   11- -000000000000--00 !---------------101
   1-- ----1---------101 --------------11-00
   1-- ---1----------101 --------------11-00
   1-- --1-----------101 --------------11-00
   1-- -1------------101 --------------11-00
   1-- 1-------------101 --------------11-00
   11- --------------001 --------------11-00
   -1- --------------100 --------------11-00
   --1 1--------------01 ----------------100
   --1 -1-------------01 ----------------100
   --1 --1------------01 ----------------100
   --1 ---1-----------01 ----------------100
   --- ----1----------01 ----------------100
   --- ---------------00 ----------------100
   -0- ---------------00 ----------------100

end truth_table
```

Figure 4-12 Transmit Counter XLT
(Sheet 2 of 2)

```
#$ TOOL NSC
#$ TITLE transmit counter
#$ TITLE
#$ DEVICE MAPL128
.i 21
.o 52
#$ PINS  19 txram:24 txack:6 framesize:5 pcin:4 pc12:11 pc11:10
pc10:9 pc9:8 pc8:7 pc7:17 pc6:18 pc5:19 pc4:20 pc3:21 pc2:22
pc1:23 irqeof0:12 moreframes:13 pc0:15
#$ NODES 5 pb1:38 pb0:37 p0:31 more:32 pb2:39
.ilb txram txack framesize pcin pc12 pc11 pc10 pc9 pc8 pc7 pc6
pc5 pc4 pc3 pc2 pc1 pb1 pb0 p0 more pb2
.ob  irqeof0.j irqeof0.k- moreframes.j moreframes.k- pc0.j-
pc0.k- pc12.j pc12.k pc11.j pc11.k pc10.j pc10.k pc9.j pc9.k
pc8.j pc8.k pc7.j pc7.k pc6.j pc6.k pc5.j pc5.k pc4.j pc4.k
pc3.j pc3.k pc2.j pc2.k pc1.j pc1.k pb1.j pb1.k pb0.j pb0.k-
p0.j- p0.k- more.j more.k- pc12.oe pc11.oe pc10.oe pc9.oe pc8.oe
pc7.oe pc6.oe pc5.oe pc4.oe pc3.oe pc2.oe pc1.oe pc0.oe pb2.reg
.type f
.phase 11111111111111111111111111111111111111111111111111111
.p 53
-0-------------00-10~~~~~~~~~~~~~~~~~~~~~~~~~~~~~0100~~~~~~~~~~~~~~~~0
----------------00-10~~~~~~~~~~~~~~~~~~~~~~~~~~~~~0100~~~~~~~~~~~~~~~~0
-1-------------00-1011~~~~~~~~~~~~~~~~~~~~~~~~~~~0100~~11~~~~~~~~~~~~~0
0---------------00--0~~~~~~~~~~~~~~~~~~~~~~~~~~~~~0111~~~~~~~~~~~~~~~~0
11-------------0001011~~~~~~~~~~~~~~~~~~~~~~~~~~110100~~11~~~~~~~~~~~~~0
11------------00001011~~~~~~~~~~~~~~~~~~~~~~~~11~~0100~~11~~~~~~~~~~~~~0
11-----------000001011~~~~~~~~~~~~~~~~~~~~~~~11~~~~0100~~11~~~~~~~~~~~~~0
11----------0000001011~~~~~~~~~~~~~~~~~~~~~~11~~~~~~0100~~11~~~~~~~~~~~~~0
11---------00000001011~~~~~~~~~~~~~~~~~~~~~11~~~~~~~~0100~~11~~~~~~~~~~~~~0
11--------000000001011~~~~~~~~~~~~~~~~~~~~11~~~~~~~~~~0100~~11~~~~~~~~~~~~~0
11-------00000000-0~~~~~~~~~~~~~~~~~~~~~11~~~~~~~~~~~~0100~~~~~~~~~~~~~~~~0
11------000000000010~~~~~~~~~~~~~~~~~~~~11~~~~~~~~~~~~~~1000~~~~~~~~~~~~~~~~0
11-----0000000000010~~~~~~~~~~~~~~~~~~~11~~~~~~~~~~~~~~~1000~~~~~~~~~~~~~~~~0
11----00000000000010~~~~~~~~~~~~~~~~~~11~~~~~~~~~~~~~~~~1000~~~~~~~~~~~~~~~~0
11---000000000000010~~~~~~~~~~~~~~~~~~11~~~~~~~~~~~~~~~~1000~~~~~~~~~~~~~~~~0
--1-1----------10-10~~~~~~~~~~~~~~~~~~~~~~~~~~~~~0100~~~~~~~~~~~~~~~~0
--------1-------10-10~~~~~~~~~~~~~~~~~~~~~~~~~~~~~0100~~~~~~~~~~~~~~~~0
11-------------10-0011~~~~~~~~~~~~~~~~~~~~~~~~~~~0100~~11~~~~~~~~~~~~~0
1---1----------10-1011~~~~~~~~~~~~~~~~~~~~~~~~~~~0100~~11~~~~~~~~~~~~~0
1----1---------10-1011~~~~~~~~~~~~~~~~~~~~~~~~~~~0100~~11~~~~~~~~~~~~~0
1-----1--------10-1011~~~~~~~~~~~~~~~~~~~~~~~~~~~0100~~11~~~~~~~~~~~~~0
1------1-------10-1011~~~~~~~~~~~~~~~~~~~~~~~~~~~0100~~11~~~~~~~~~~~~~0
--1----1--------10-10~~~~~~~~~~~~~~~~~~~~~~~~~~~~~0100~~~~~~~~~~~~~~~~0
--1---1---------10-10~~~~~~~~~~~~~~~~~~~~~~~~~~~~~0100~~~~~~~~~~~~~~~~0
```

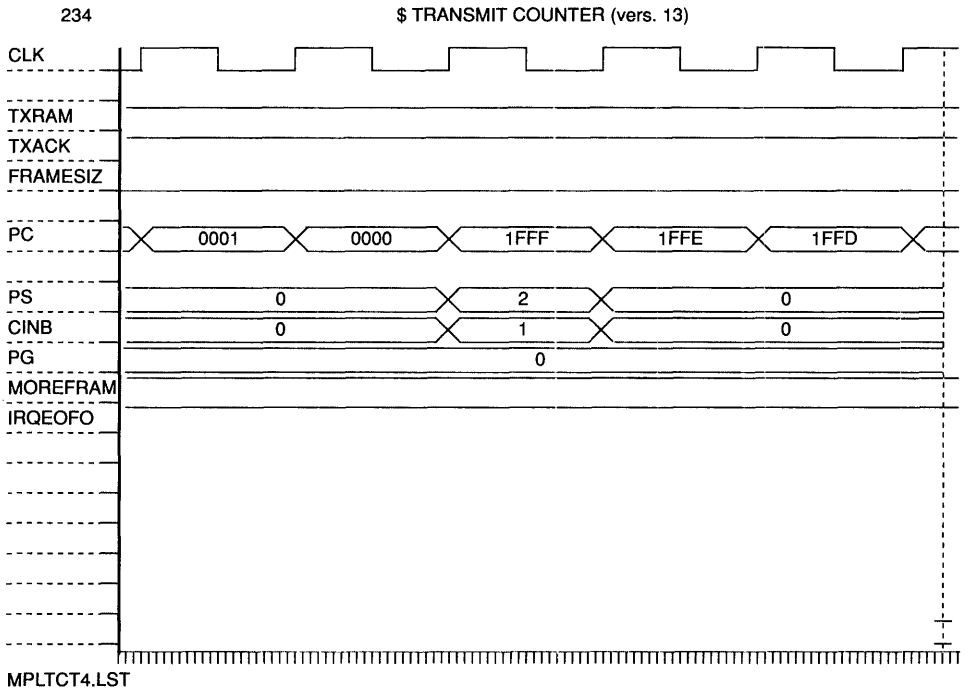
Figure 4-13 Output from Transmit Counter
(Sheet 1 of 2)

Figure 4-13
(Sheet 2 of 2)



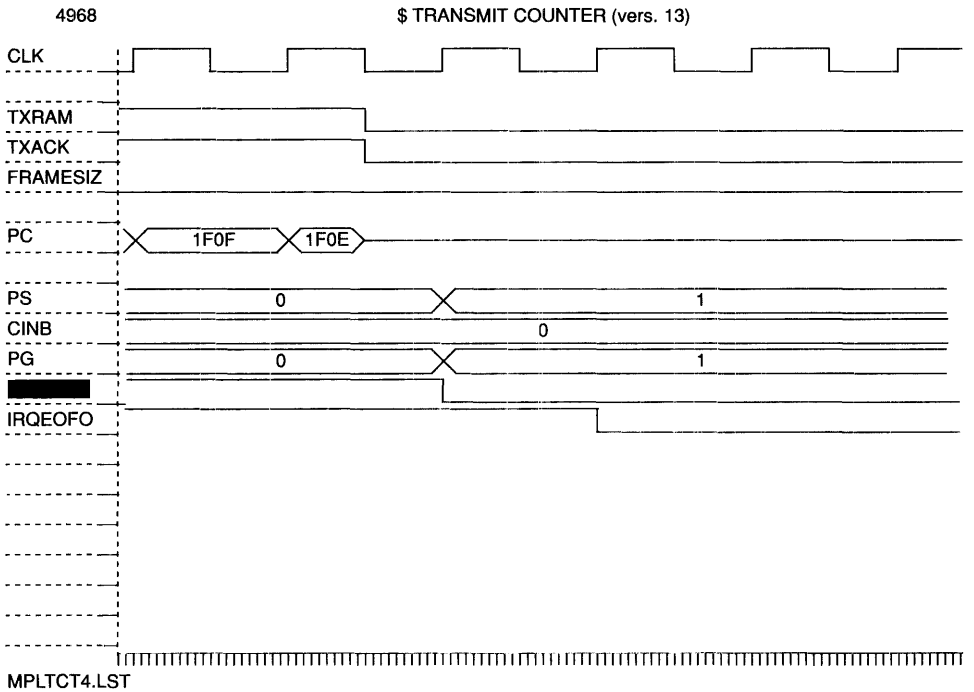
TSP-MAPL-14

Figure 4-14 Timing Diagram



TSP-MAPL-15

Figure 4-15 Timing Diagram



TSP-MAPL-16

Figure 4-16 Timing Diagram

Signal Description

Inputs

FRAMESIZE	Determines the maximum size frame that may be transmitted. 0: 512 bytes 1: 8192 bytes This is used by the MAPL design to determine the boundary crossing that signifies the end of a frame.
TXACK	Transmit acknowledge output from the NSC FDDI BMAC (DP83261) Indicates that the transmitter is ready for the next data byte.
TXRAM~	Enables the MAPL design to address the Transmit RAM.
PC_ADDR(12:0)	These are the address inputs to the MAPL design. The address lines can be loaded and then incremented to address the Transmit RAM.

Outputs

MOREFRAMES	MOREFRAMES is asserted whenever there are more 512 byte frames in the Tx_RAM to be transmitted.
IRQEOF0	Request end of frame. Indicates that the data ready to be transmitted is the last data byte when asserted. Normally, this is the last byte of the INFO field of the frame.
ADDROUT(12:0)	These are the latched PC_ADDR signals that address the Tx_RAM. The MAPL design takes the PC_ADDR, loads it, and then increments through the addresses until the Tx_RAM is empty.

4.7 A MAPL Divide Down Counter with Specific Timing Control Options

Introduction

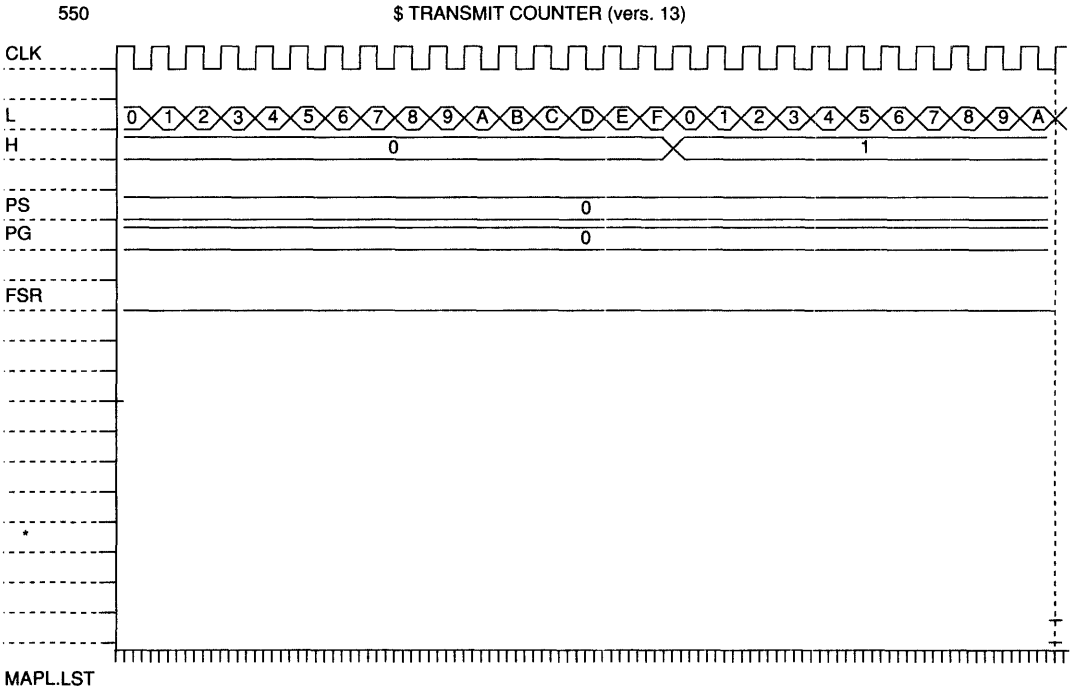
This application note demonstrates the use of a MAPL128 in a PC add-in FAX/modem card. The MAPL will be used to generate a repetitive 9.6 KHz clock output that can be advanced or delayed by the user.

Description Of Design

Because of the MAPL128's register intensive nature, four simple synchronous state machines can be integrated into one MAPL128 device for this application. The function of the MAPL128 is to divide an input clock of frequency 1.536 MHz down by 160 (8 state bits required) to achieve a repetitive output clock of frequency 9.6 KHz. Besides simply dividing an input clock, the MAPL128 allows specific user programmable control. A user can choose to advance, (cause the output pulse to occur 4 clock cycles early), or delay, (cause the output pulse to occur 4 clock cycles late), the output through the use of two control bits named advance and delay. For example, if a user wishes to advance the output, he would write the control bit advance and instead of a divide by 160 output, we have a divide by 156 output. In other words, the output occurs 4x1.536 MHz or 2.6us earlier than expected. This programmable feature is incorporated in two simple synchronous state machines that fit very well into the MAPL128.

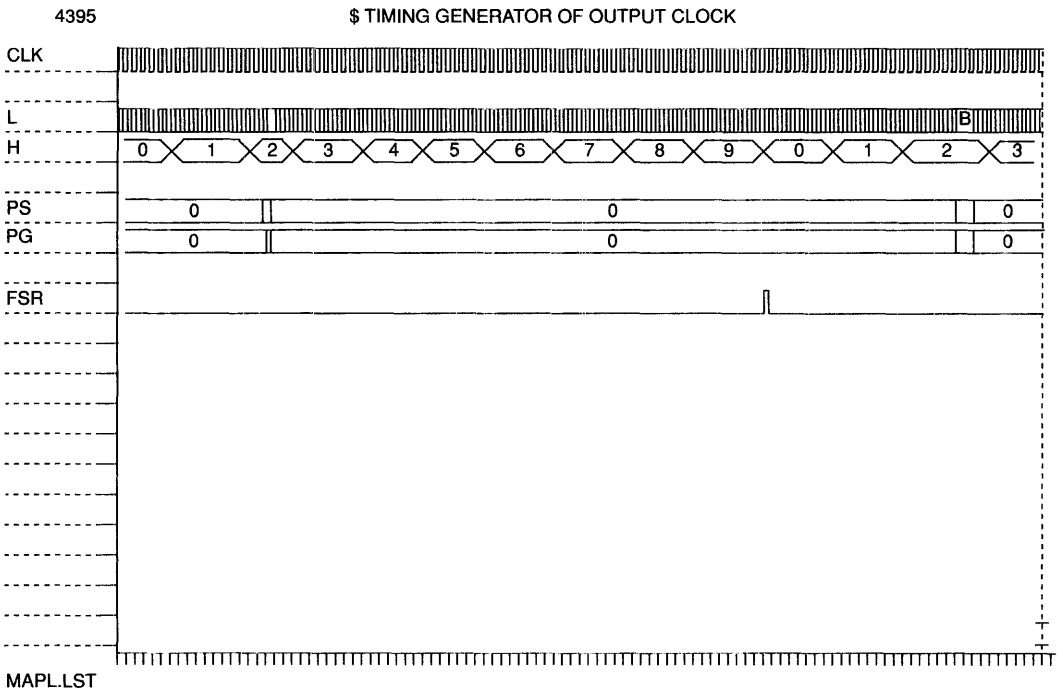
The motivation for using the MAPL128 is twofold. Because the MAPL128 replaces three PAL devices, it saves both board space and chip count.

Let's analyze the state machines that make up the MAPL128 in more detail: We achieve the divide by 160 function by implementing a low counter divide of 16 and a high counter divide of 10. Notice the timing diagram of Figure 4-17 as every sixteenth MCLK, the H state changes and upon the transition of Hstate 9 -> 0, the output pulse (Figure 4-18) is generated, producing a repetition of 9.6 KHz (1/160x1.536 MHz).



TSP-MAPL-18

Figure 4-17 Timing Diagram



TSP-MAPL-19

Figure 4-18 Timing Diagram

To advance the output, the user must write the control bit advance. Note the state diagram in Figure 4-19 to see how the advance input causes a skipping of four states, thus achieving the necessary four clock cycle advance. Note also in Figure 4-20 how a "cancel advance" signal is generated by the simple advance state machine. Cancel advance is necessary to clear the advance control bit written by the user. Without the cancel signal, the main counter state machine will keep skipping. Likewise, a user can realize a delaying of the output by writing the control bit delay. Note in Figure 4-21 how "delay" causes a repeat of the same state, eventually staying in the same state four extra clocks to net a result output four clock cycles later. See how the delay feature repeats the state four times before it issues the CANCEL_DEL pulse which clears the delay control bit written by the user and allows the main counter state machine to continue.

The implementation of this design in a .OPL file can be seen in Figure 4-22. Since there are a few state machines in this design, the truth table function proved to be the simplest solution. The first section of the truth table implements the normal state transitions for the L states and the H states. This portion of the truth table also implements transitions from the normal state to advance or delay functions. The second portion of the truth table implements the advance function while the third portion implements the delay function.

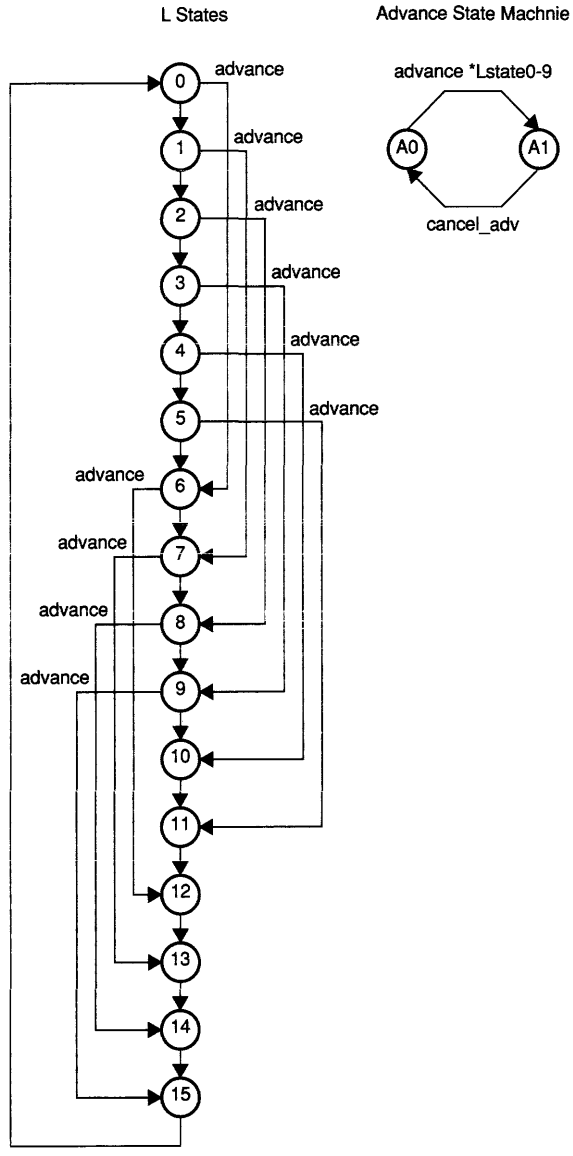
Signal Description

Inputs

MCLK	Input clock of frequency 1.536 MHz.
ADVANCE	User generated control bit used to cause output clock to occur 4 MCLKs earlier.
DELAY	User generated control bit used to cause output clock to occur 4 MCLKs later.

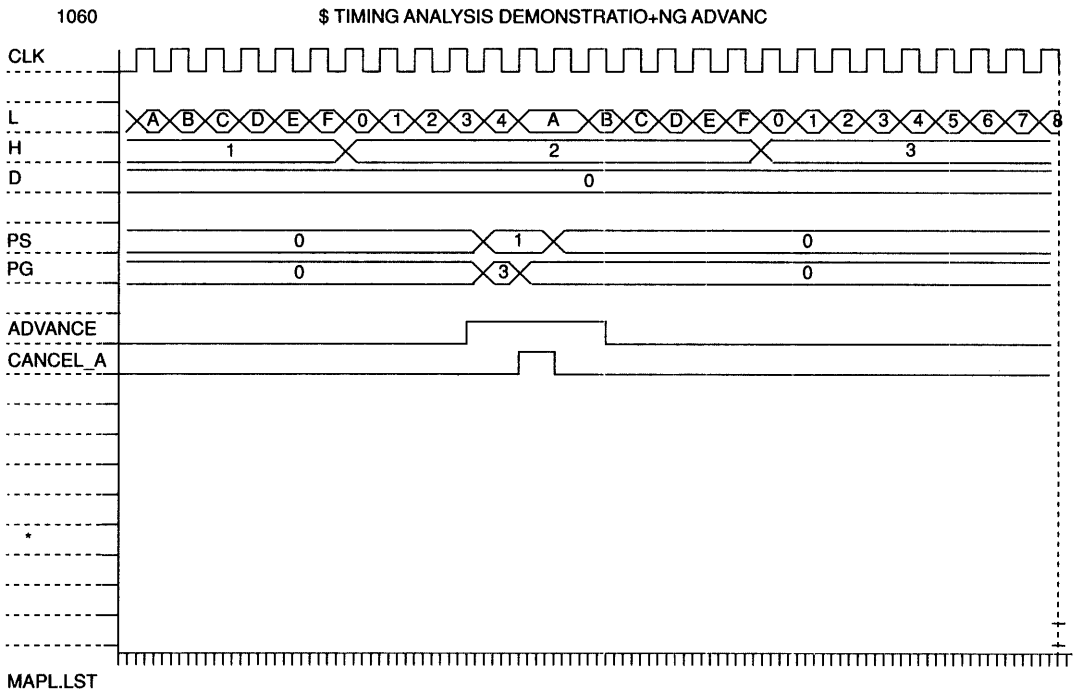
Outputs

FSR	Output clock of frequency 9.6 KHz.
CANCEL_ADV	Signal generated by MAPL to reset advance control bit.
CANCEL_DEL	Signal generated by MAPL to reset delay control bit.



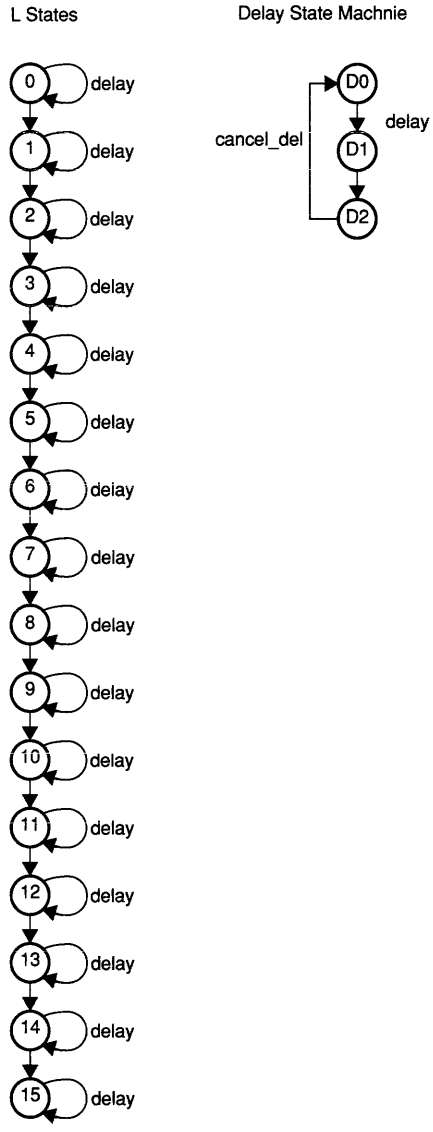
TSP-MAPL-20

Figure 4-19 Advance State Machine



TSP-MAPL-21

Figure 4-20 Timing Diagram



TSP-MAPL-22

Figure 4-21 Delay State Machine

```
Begin Header
   Divide Down Counter with Specific Timing Control Options
End Header

Begin Definition
   Inputs
      rst, advance, delay ;
   Feedbacks (JK, HOLD)
      h3, h2, h1, h0, l3, l2, l1, l0, d2, d1, d0;
   Feedbacks (JK,RST)
      a0;
   Feedbacks (JK,RST,BURIED)
      g0, g1;
   Outputs
      cancel_adv, cancel_del, fsr ;
End Definition

Begin Equation
   global.re = rst;
End Equation

Begin Truth_Table
   ttin  advance, delay, h3, h2, h1, h0, l3, l2, l1, l0, a0, d2,
      d1, d0, g1, g0;
   ttout h3, h2, h1, h0, l3, l2, l1, l0, a0, d2, d1, d0,
      cancel_adv, cancel_del, fsr, g1, g0;

      -0-------- - ---00     -------! - --- --000
      -0-------1 - ---00     ------!- - --- --000
      -0------11 - ---00     -----!-- - --- --000
      -0-----111 - ---00     ----!--- - --- --000
      -0----1111 - ---00     ---!---- - --- --000
      -000011111 - ---00     0010---- - --- --000
      -00-101111 - ---00     0?11---- - --- --000
      -000111111 - ---00     0100---- - --- --000
      -001001111 - ---00     0101---- - --- --000
      -001011111 - ---00     0110---- - --- --000
      -001111111 - ---00     1000---- - --- --000
      -010001111 - ---00     1001---- - --- --000
      -010011111 - ---00     0000---- - --- --100
      10----0--- - ---00     -------- - --- --011
      10----100- - ---00     -------- - --- --011
      01-------- - ---00     -------- - 000 ---10
```

Figure 4-22 Divide Down Counter
(Sheet 1 of 2)

```
      10----0000 0 ---11      ----0110 1 --- 1-000
      10----0001 0 ---11      ----0111 1 --- 1-000
      10----0010 0 ---11      ----1000 1 --- 1-000
      10----0011 0 ---11      ----1001 1 --- 1-000
      10----0100 0 ---11      ----1010 1 --- 1-000
      10----0101 0 ---11      ----1011 1 --- 1-000
      10----0110 0 ---11      ----1100 1 --- 1-000
      10----0111 0 ---11      ----1101 1 --- 1-000
      10----1000 0 ---11      ----1110 1 --- 1-000
      10----1001 0 ---11      ----1111 1 --- 1-000
      10----1010 0 ---11      ----1011 1 --- 1-000
      10----1011 0 ---11      ----1100 0 --- 0-000
      10----1100 0 ---11      ----1101 0 --- 0-000
      10----1101 0 ---11      ----1110 0 --- 0-000
      10----1110 0 ---11      ----1111 0 --- 0-000
      10----1111 0 ---11      ----0000 0 --- 0-000

      01-------- - 00010      -------- - 001 -0010
      01-------- - 00110      -------- - 011 -0010
      01-------- - 01110      -------- - 000 -1000

End Truth_Table
```

Figure 4-22
(Sheet 2 of 2)

4.8 Bus Transaction and DMA Controller

This application demonstrates the use of the MAPL128 for bus state machine control. In this design, the MAPL128 provides bus access and DMA control for an I/O board on an asynchronous bus (Futurebus+). Although the design is fairly simple, the resulting state machine demonstrates flexibility and power of the OPAL command language. Less than half of the product terms of the MAPL128 are used in this example, so there is room for substantial increases in design sophistication.

The MAPL128 state machine controls accesses from the system bus to the local DRAM, from the local CPU to the system bus, and block data transfers between local DRAM and the system bus. The DMA controller is a discrete design, programmed by the CPU and enabled by the state machine. It consists of four sub-blocks:

1. The local address counter is programmed by the CPU before each DMA operation with the start address of the data block in DRAM. This can be either the source or destination of the transfer, depending on whether a bus write or read has been programmed.

2. The bus address counter is also programmed by the local CPU with the start address of the data block on Futurebus. Once again, this is independent of the direction of data transfer.

3. The transfer counter determines the total size of the DMA transfer. This is programmed by the local CPU at the start of each transfer. The local CPU can read the contents of the counter in the event of an error or unexpected DMA termination condition. Because this is an asynchronous bus, this counter must be clocked asynchronously as the DMA transfer progresses, and could not be implemented in the MAPL128.

4. The packet counter determines the number of words in each bus transaction, breaking up the DMA transfer into blocks that the bus can handle. The 8-bit counter allows packet sizes of up to 256 words (1 Kbytes on a 32-bit bus) in a single bus transaction. This is programmed by the lo-

cal CPU as necessary; the counter is automatically reloaded with the last programmed value before each bus transfer.

The MAPL128 state machine takes requests from the system bus and generates accesses into local DRAM, takes requests from the CPU and generates accesses out to the system bus, and, after the CPU has loaded the appropriate DMA controller registers, generates accesses to both the DRAM and the system bus. It enables the address latches for each type of access. It monitors the transfer and packet counters, so that multiple packets are generated until all the data is transferred. The actual data transfer is controlled by a separate asynchronous state machine.

The state machine implemented by the MAPL128 contains three "subroutines" for the three types of accesses supported. TheMAPL128 waits in the idle state until a request occurs. The state machine language uses the if/else if/else syntax to prioritize simultaneous requests. BUS_ADDR_HIT is generated by the system bus address decoder, CPU_ADDR_HIT is from the decoder on the local bus, and CPU_GO signals that the DMA has been programmed by the CPU. After a system bus request, the slave subroutine waits for a bus grant and free local bus before beginning the access and enabling data transfer. The machine supports locked transactions, which require that the local bus not be released between transfers. The master (CPU access) subroutine is almost identical to the slave subroutine. The specific signals monitored and generated by the MAPL128 can be easily modified for any type of CPU or bus interface.

The DMA state machine first requests the system bus, then the local bus before performing accesses. It will continue to generate packets until all the data has been transferred and both counters are zero. (The packet counter gets reloaded when it reaches zero, until the transfer counter also reaches zero).

The DMA subroutine waits for the CPU to signal, via an external I/O control register bit, that the DMA registers described above have been programmed and the transfer is ready to go. It requests first the system bus, then the local bus. Once both buses are available, it enables the addresses onto both the local and system buses and starts the asynchronous bus transfer state machine. With each

data transfer, the transfer and packet counters are decremented. The state machine will give up the local and system busses between transfers in order to allow other devices access to them. Once the transfer counter reaches zero, the DMA operation is complete. The DMA state machine checks to ensure that the transfer has completed successfully, and interrupts the CPU.

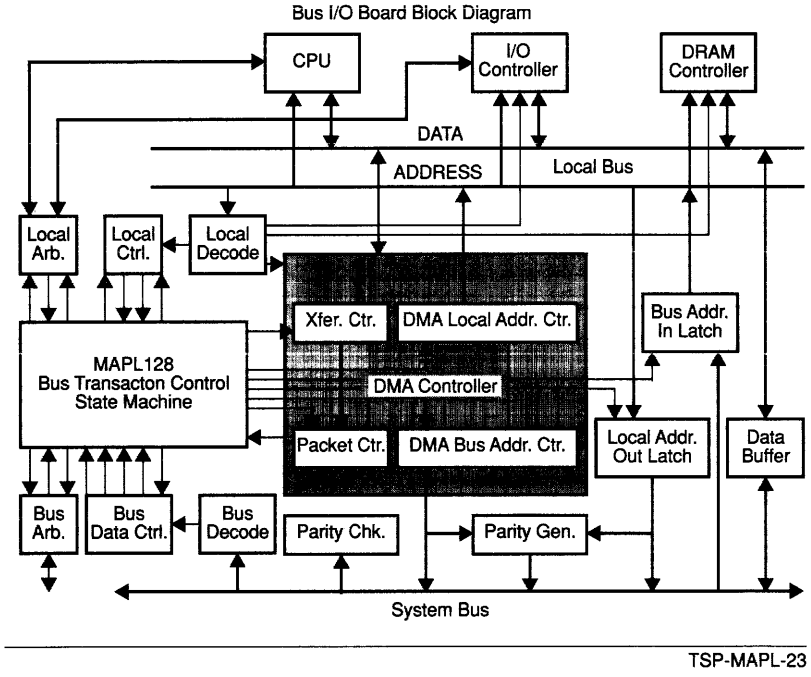
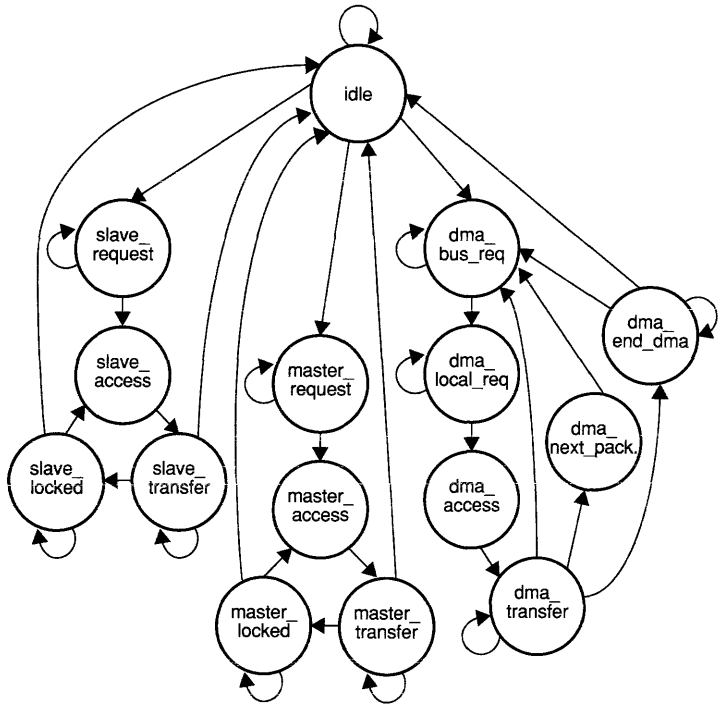
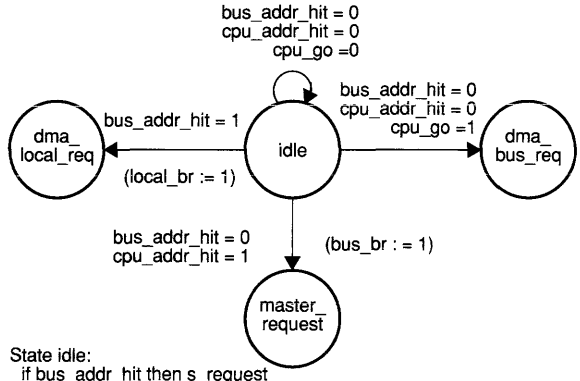


Figure 4-23 Bus I/O Board Block Diagram



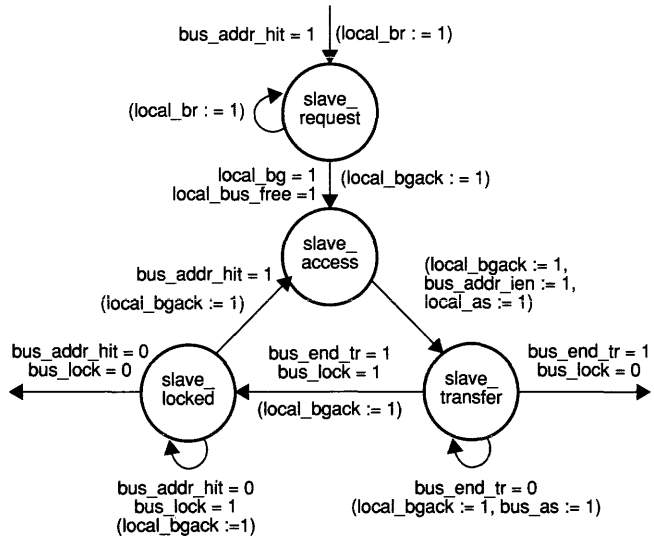
TSP-MAPL-28

Figure 4-24 Bus Control State Machine



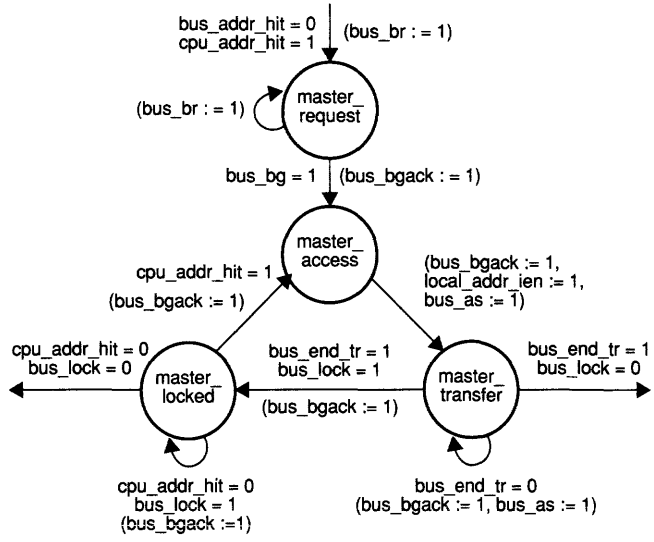
TSP-MAPL-25

Figure 4-25 Bus Control State Machine (Idle State)



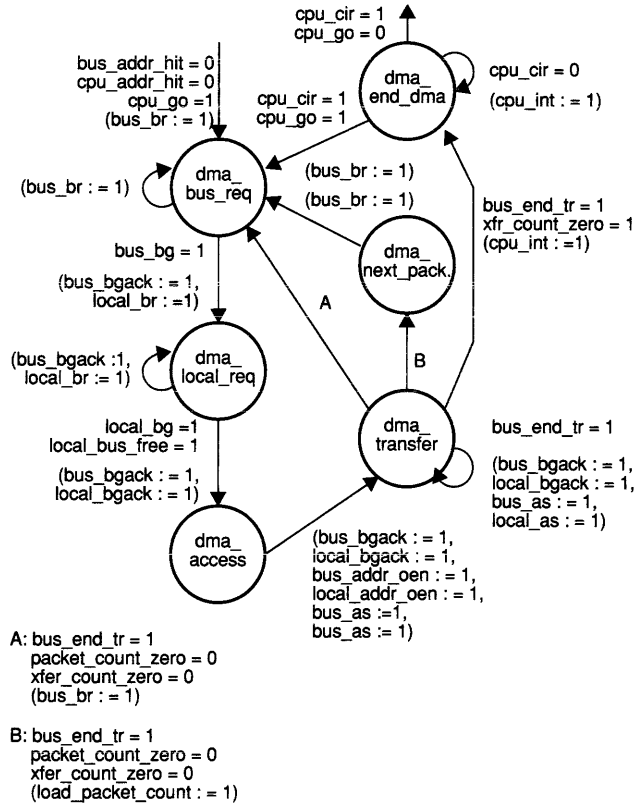
TSP-MAPL-27

Figure 4-26 Bus Control State Machine (Slave)



TSP-MAPL-26

Figure 4-27 Bus Control State Machine (Master)



TSP-MAPL-24

Figure 4-28 Bus Control State Machine (DMA)

```
begin header

Synchronous Bus Control State Machine - Rev 0.1, Dave Hawley

end header

begin definition

{

Inputs:

CPU_GO                  Signal from the CPU to begin DMA transfers
CPU_CLR                 Signal from the CPU to clear DMA interrupt
LOCAL_BG                Bus grant from local arbiter
LOCAL_BUS_FREE          Bus free status on local bus
CPU_ADDR_HIT            Local decoder recognizes CPU access to sys-
                        tem bus
PACKET_COUNT_ZERO       DMA data transfer counter equals zero
XFER_COUNT_ZERO         DMA data transfer counter equals zero
BUS_ADDR_HIT            Bus decoder recognizes system access to
                        DRAM
BUS_BG                  System bus grant
BUS_END_TR              System bus end of data transfer
BUS_LOCK                System bus lock
BUS_ERROR               System bus transfer error

Outputs:

CPU_INT                 DMA complete interrupt signal to CPU
LOCAL_BR                Local bus request from DMA
LOCAL_BGACK             Local bus grant acknowledge
LOCAL_AS                Local bus address strobe
BUS_ADDR_IEN            Latch enable for system bus address to lo-
                        cal bus
LOCAL_ADDR_OEN          Latch enable for DMA address to local bus
LOAD_PACKET_COUNT       Load packet transfer size counter
BUS_BR                  System bus request
BUS_BGACK               System bus grant acknowledge
BUS_AS                  System bus address strobe
LOCAL_ADDR_IEN          Latch enable for CPU address to system bus
BUS_ADDR_OEN            Latch enable for DMA address to system bus
DMA_ERROR               DMA transfer error
}

inputs
   cpu_go, cpu_clr, local_bg, local_bus_free, cpu_addr_hit,
```

```
   packet_count_zero, xfer_count_zero, bus_addr_hit,
   bus_bg, bus_end_tr, bus_lock, bus_error;

outputs
   cpu_int, local_br, local_bgack, local_as,
   bus_addr_ien, local_addr_oen, load_packet_count,
   bus_br, bus_bgack, bus_as,
   local_addr_ien, bus_addr_oen, dma_error;

statebits (buried)
   s3,s2,s1,s0;

end definition

begin state_diagram

{
|  When system bus address hit, request local bus
|  When CPU address hit, request System Bus
|  When CPU says go, start DMA transfer (request System Bus,
|     then local bus)
}
state idle :
   if bus_addr_hit then s_request
      with local_br := 1;
      endwith
   else
      if cpu_addr_hit then m_request
         with bus_br := 1;
         endwith
      else
         if cpu_go then d_bus_req
            with bus_br := 1;
            endwith
         else idle;
{
*** Slave State Machine ***
}
{
|  Wait for local bus free (bus grant, all other signals
|  released) (assert BGACK before release of BR)
}
state s_request :
   if local_bg * local_bus_free then s_access
      with local_bgack := 1;
           local_br := 0;
      endwith
```

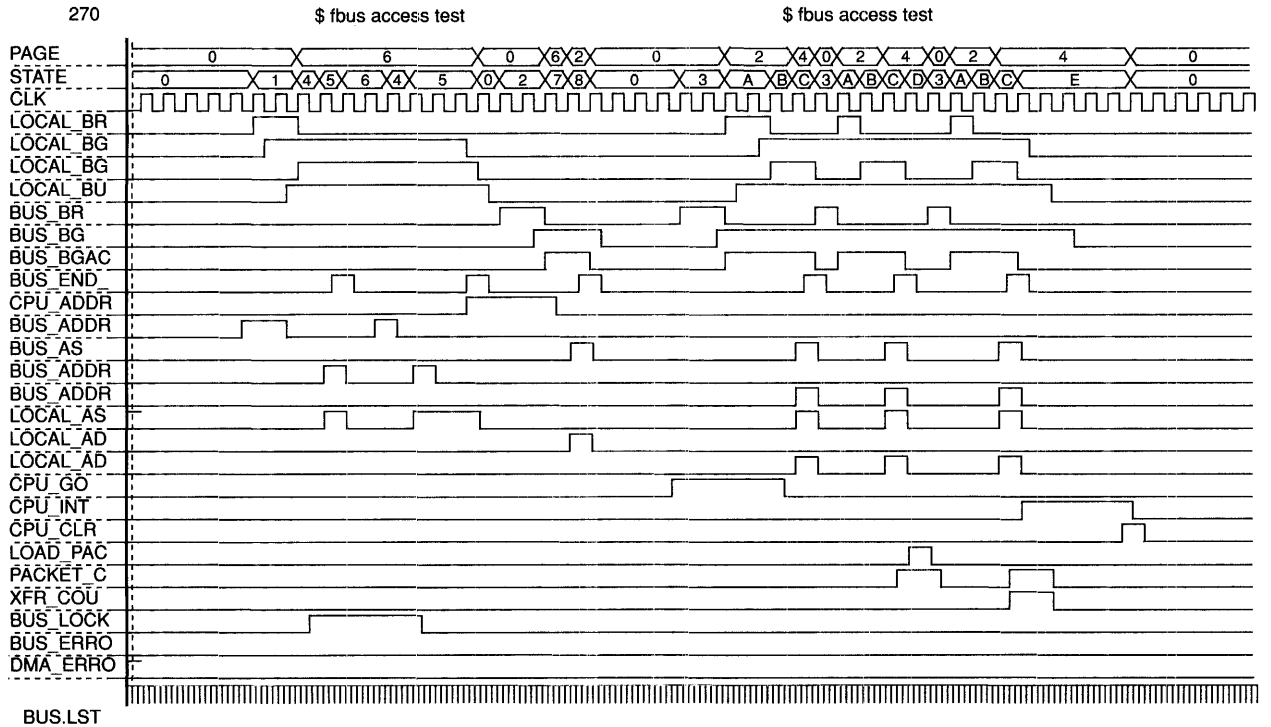
```
   else s_request
      with local_br := 1;
      endwith;
{
|  Begin bus transaction
|  (assert address enable before AS)
}
state s_access :
   goto s_transfer
      with local_bgack := 1;
         bus_addr_ien := 1;
         local_as := 1;
      endwith;
{
|  Can release the address enable immediately (latched by
|     DRAM controller)
|  When transaction complete, release AS
|  If no lock, release local bus, otherwise enter locked state
}
state s_transfer :
   case bus_end_tr * /bus_lock : idle;
         bus_end_tr * bus_lock : s_locked
            with local_bgack := 1;
            endwith;
         /bus_end_tr : s_transfer
            with local_bgack := 1;
               bus_addr_ien := 0;
               local_as := 1;
            endwith;
   endcase;
{
|  In locked state, wait for new address hit or lock release
}
state s_locked :
   case bus_addr_hit : s_access
         with local_bgack := 1;
         endwith;
      /bus_addr_hit * /bus_lock : idle;
      /bus_addr_hit * bus_lock : s_locked
         with local_bgack := 1;
         endwith;
   endcase;
{
*** Master State Machine ***
}
{
| Wait for system bus grant
```

```
| (assert BGACK before release of BR)
}
state m_request :
   if bus_bg then m_access
      with bus_bgack := 1;
           bus_br := 0;
      endwith
   else m_request
      with bus_br := 1;
      endwith;
{
| Begin bus transaction
| (assert address enable before AS)
}
state m_access :
   goto m_transfer
      with bus_bgack := 1;
         local_addr_ien := 1;
         bus_as := 1;
      endwith;
{
| Can release the address enable immediately
| When transaction complete, release AS
| If no lock, release System Bus, otherwise enter locked state
}
state m_transfer :
   case bus_end_tr * /bus_lock : idle;
        bus_end_tr * bus_lock : m_locked
           with bus_bgack := 1;
           endwith;
        /bus_end_tr : m_transfer
           with bus_bgack := 1;
              local_addr_ien := 0;
              bus_as := 1;
           endwith;
   endcase;
{
| In locked state, wait for new address hit or lock release
}
state m_locked :
   case cpu_addr_hit : s_access
        with bus_bgack := 1;
        endwith;
      /cpu_addr_hit * /bus_lock : idle;
      /cpu_addr_hit * bus_lock : s_locked
        with bus_bgack := 1;
        endwith;
```

```
      endcase;
{
*** DMA State Machine ***
}
{
| Wait for System Bus grant, then request local bus
| (assert BGACK before release of BR)
}
state d_bus_req :
   if bus_bg then d_local_req
      with bus_bgack := 1;
         bus_br := 0;
         local_br := 1;
      endwith
   else d_bus_req
      with bus_br := 1;
      endwith;
{
| Wait for local bus free (bus grant, all other signals released)
| (assert BGACK before release of BR)
}
state d_local_req :
   if local_bg * local_bus_free then d_access
      with bus_bgack := 1;
            local_bgack := 1;
            local_br := 0;
      endwith
   else d_local_req
      with bus_bgack := 1;
         local_br := 1;
      endwith;
{
| Begin DMA transaction
| (assert address enables before AS)
}
state d_access :
   goto d_transfer
      with bus_bgack := 1;
            local_bgack := 1;
            bus_addr_oen := 1;
            local_addr_oen := 1;
            bus_as := 1;
            local_as := 1;
      endwith;
{
| Can release the address enables immediately
| When transaction complete, release AS
```

```
| Interrupt CPU, reload packet counter, or re-request bus,
|   depending on packet and transfer counter states
}
state d_transfer :
   case bus_end_tr * /bus_error * /packet_count_zero *
              /xfer_count_zero :
         d_bus_req
            with bus_br := 1;
            endwith;
         bus_end_tr * /bus_error * packet_count_zero *
              /xfer_count_zero :
         d_next_packet
            with load_packet_count := 1;
            endwith;
         bus_end_tr * /bus_error * packet_count_zero *
              xfer_count_zero :
         d_end_dma
            with cpu_int := 1;
            endwith;
         bus_end_tr * /bus_error * /packet_count_zero *
              xfer_count_zero :
         d_end_dma
            with cpu_int := 1;
               dma_error := 1;
            endwith;
         /bus_end_tr : d _transfer
            with bus_bgack := 1;
               local_bgack := 1;
               bus_addr_oen := 0;
               local_addr_oen := 0;
               bus_as := 1;
               local_as := 1;
            endwith;
   endcase;
{
| Transfer count <> 0, so reload packet counter and continue
}
state d_next_packet :
   goto d_bus_req
      with load_packet_count := 0;
            bus_br := 1;
      endwith;
{
| Error or transfer count = 0 so interrupt CPU and
| wait for clear interrupt
}
{
```

```
state d_end_dma :
   case cpu_clr * /cpu_go : idle;
         cpu_clr * cpu_go : d_bus_req
            with bus_br :- 1;
            endwith;
         /cpu_clr * cpu_go : d_end_dma
            with cpu_int := 1;
               dma_error := 1;
            endwith;
         /cpu_clr * /cpu_go : d_end_dma
            with cpu_int := 1;
            endwith;
   endcase;

end state_diagram
```



TSP-MAPL-29

Figure 4-29 Timing Diagram

4.9 A MAPL PC Interface Module for the FDDI MAC Layer Evaluation Board

Introduction

This application uses the MAPL144, Multiple Array Programmable Logic device, as a PC Interface Module. The MAPL144 provides access to the transmit and receive RAMs in addition to board registers, and the BMAC peripheral chip. Since the MAPL144 is a sequential device, the decode logic is based on a state-machine which controls accesses from the system bus. OPAL software is used to compile the high-level .OPL file into a JEDEC map which can be used with Viewlogic or OPALSIM to test and verify the design. It is assumed that the reader is already familiar with the FDDI MAC Layer Evaluation Board.

Description of Design

Figure 4-30 gives a simple block diagram of the FDDI MAC Board Data Path, while Figure 4-31 gives a block diagram of the Control Logic. The function of the PC Interface Module is to interface the FDDI MAC Layer Evaluation Board to the PC host. This block features a 20-bit address bus for flexible memory map placement. In addition to the address bus, five PC bus control signals as well as a couple of board interface signals are included.

The PC Interface Module provides the chip selects and intermediate signals needed for memory select. The board registers MODE, STATUS, and FUNCTION are all controlled by the PC Interface Module. The peripheral BMAC chip is also controlled by the PC Interface with the signal BMACSEL. The intermediate signals generated by the PC Interface for memory control are MEMSEL and ACK3. Figure 4-32 shows the address mapping for the specific functions on the board.

The PC interface currently used on the FDDI MAC Layer Evaluation Board utilizes four GAL20V8's. The entry files for these four GAL devices were obtained using Boolean equations. Three of the GAL devices can be integrated into the MAPL144. The fourth GAL device provides combinatorial access of the memories on the board.

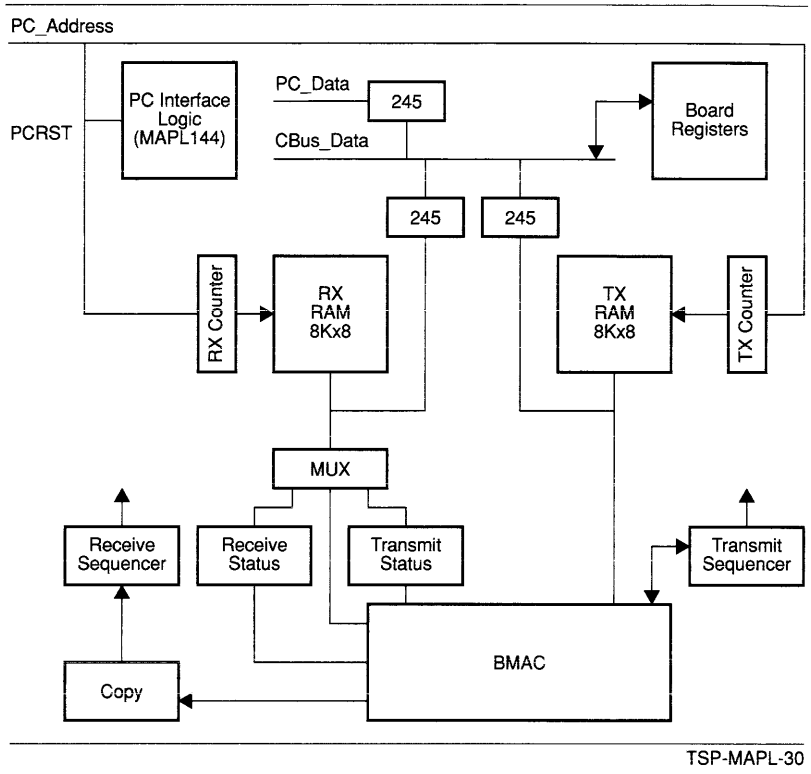
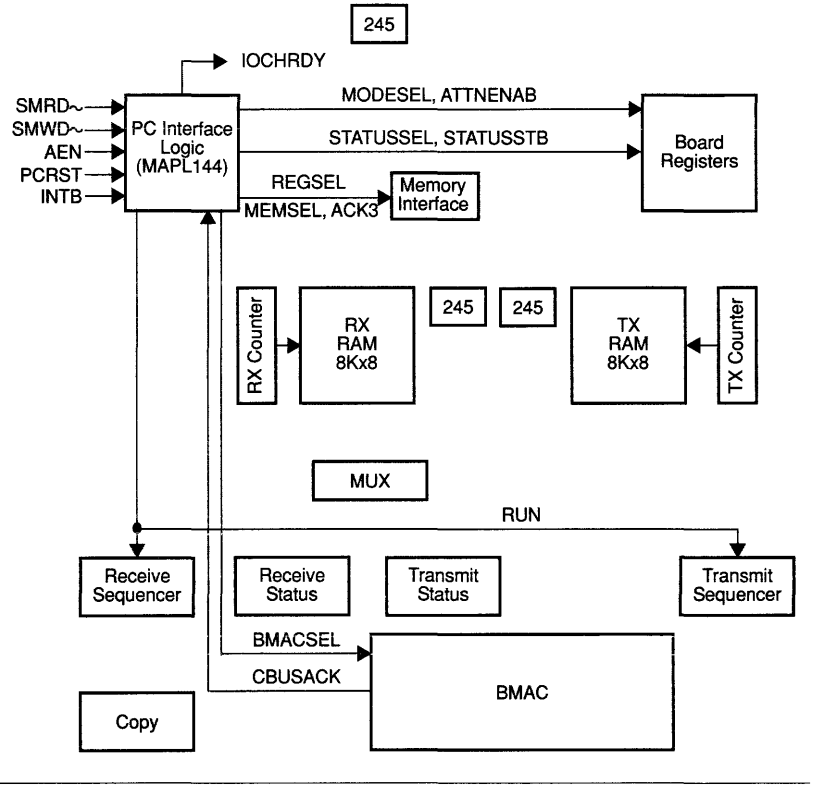


Figure 4-30 FDDI MAC Board Block Diagram Data Path

When the Boolean equations for the three GAL devices are combined into one .OPL file, the logic will not compile into the device because of logic partitioning problems. In other words, it is unable to allocate logic to different pages when it is specified as being on one page. The PC interface was redesigned based on a state machine so as to split up the decode logic. Most of the decode logic in the Boolean equations was based on access feedback terms. These access feedback terms could be incorporated into a state machine. This access control state machine is illustrated in Figure 4-33. Now, each of the eight states can be placed on one page of the MAPL144, thus partitioning the decode logic.

Incorporating the state machine into the .OPL file is now trivial. Figure 4-34 shows the final .OPL file for this design. Notice how the page bits are defined as state bits in the definition block. This allows the designer to free-up three pins for designs that are I/O limited. Figure 4-35 shows the output from the fitter. In this particular design, manual paging was done by using the page bits as



TSP-MAPL-31

Figure 4-31 FDDI MAC Board Block Diagram Control Logic

state bits. This way, the designer has the freedom to partition the design the way he wants it to be partitioned.

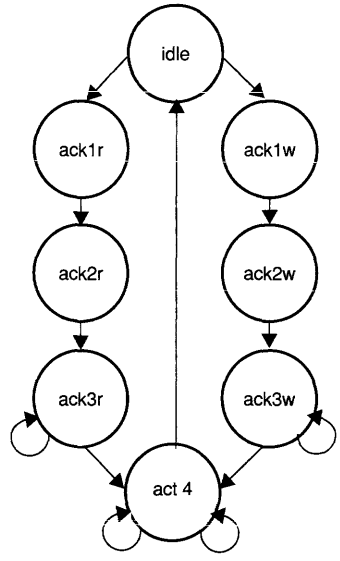
Signal Description

Inputs

SMWR~	System write.
SMRD~	System read.
AEN~	Address Enable.
PCRST	PC Reset.
CBUSACK	CBUS Acknowledge from BMAC peripheral.
INTB	Inverse Interrupt Signal.

	PC19	PC18	PC17	PC16	PC15	PC14	PC11	PC10	PC9	PC8	PC7	PC6
REGSEL	1	1	0	1	0	-	-	-	-	-	-	-
MEMSEL	1	1	0	1	1	-	-	-	-	-	-	-
RECEIVE	-	-	-	-	-	1	-	-	-	-	-	-
TRANSMIT	-	-	-	-	-	0	-	-	-	-	-	-
IOCHCTL	1	1	0	1	-	-	-	-	-	-	-	-
ACK1	1	1	0	1	-	-	-	-	-	-	-	-
CBUSACCESS	-	-	-	-	-	-	0	0	1	0	-	-
BMACSEL	-	-	-	-	-	-	0	0	1	0	-	-
BOARDREG	-	-	-	-	-	-	0	0	1	1	-	-
MODESEL	-	-	-	-	-	-	-	-	-	-	0	0
ATTNENAB	-	-	-	-	-	-	-	-	-	-	0	1
RUN	-	-	-	-	-	-	-	-	-	-	1	1

Figure 4-32 PC Interface Logic Signal Description for MP144



TSP-MAPL-32

Figure 4-33 PC Interface Access Control State Machine

```
begin header
   pc interface (state machine)
end header

begin definition
   device MAPL144;

   inputs
      aen,smwr,smrd,
      pc19,pc18,pc17,pc16,pc15,pc11,pc10,pc9,pc8,
      pc7,pc6,pc5,pcrst,cbusack,intb;
   outputs (JK,RST)
      statussel,modesel,attnenab,run,bmacsel,ack3,memsel,
      regsel,statusstb,int,iochrdy;
   statebits (DE,RST,BURIED)
      pb2=47,pb1=46,pb0=45;
   STATE_NAME
      IDLE=^b000,ACK1W=^b001,ACK1R=^b100,ACK2W=^b010,
      ACK2R=^b101,ACK3W=^b011,ACK3R=^b110,ACK4RW=^b111;

end definition

begin equation

   int := /intb;

end equation

begin state_diagram
   state IDLE :
      CASE
      /smrd*/aen*pc19*pc18*/pc17*pc16 : ACK1R
         with /iochrdy := 1; endwith;
      /smwr*/aen*pc19*pc18*/pc17*pc16 : ACK1W
         with /iochrdy := 1; endwith;
      ENDCASE;

   state ACK1W :
      goto ACK2W
      with
         modesel := /pc15*/pc11*/pc10*pc9*pc8*/pc7*/pc6;
```

Figure 4-34 PC Interface .OPL File
(Sheet 1 of 4)

```
      attnenab := /pc15*/pc11*/pc10*pc9*pc8*/pc7*pc6;
      statussel := (pc7*/pc6*/pc5*/pc15*/pc11*/
         pc10*pc9*pc8);
      statusstb := pc7*/pc6*/pc5;
      run := pc7*pc6*/pc15*/pc11*/pc10*pc9*pc8+pcrst;
      bmacsel := /pc15*/pc11*/pc10*pc9*/pc8;
      regsel := /pc15;
      memsel := pc15;
      /iochrdy := 1;
   endwith;

state ACK1R :
   goto ACK2R
   with
      modesel := /pc15*/pc11*/pc10*pc9*pc8*/pc7*/pc6;
      attnenab := /pc15*/pc11*/pc10*pc9*pc8*/pc7*pc6;
      statussel := (pc7*/pc6*/pc5*/pc15*/pc11*/
         pc10*pc9*pc8);
      statusstb := pc7*/pc6*/pc5;
      run := pc7*pc6*/pc15*/pc11*/pc10*pc9*pc8+pcrst;
      bmacsel := /pc15*/pc11*/pc10*pc9*/pc8;
      regsel := /pc15;
      memsel := pc15;
      /iochrdy := 1;
   endwith;

state ACK2W :
   goto ACK3W
   with
      modesel := ?;
      attnenab := ?;
      statussel := ?;
      statusstb := ?;
      run := ?;
      bmacsel := ?;
      regsel := ?;
      memsel := ?;
      ack3 := 1;
      /iochrdy := 1;
   endwith;
```

Figure 4-34 PC Interface .OPL File
(Sheet 2 of 4)

```
state ACK2R :
   goto ACK3R
   with
      modesel := ?;
      attnenab := ?;
      statussel := ?;
      statusstb := ?;
      run := ?;
      bmacsel := ?;
      regsel := ?;
      memsel := ?;
      ack3 := 1;
      /iochrdy := 1;
   endwith;

state ACK3W :
   if (pc15+pc11+pc10+/pc9+pc8)+
                (/pc15*/pc11*/pc10*pc9*/pc8*/cbusack)
   then ACK4RW
   else ACK3W
   with
      modesel := ?;
      attnenab := ?;
      statussel := ?;
      statusstb := ?;
      run := ?;
      bmacsel := ?;
      regsel := ?;
      memsel := ?;
      ack3 := 1;
      /iochrdy := 1;
   endwith;
```

Figure 4-34 PC Interface .OPL File
(Sheet 3 of 4)

```
   state ACK3R :
      if (pc15+pc11+pc10+/pc9+pc8)+
                  (/pc15*/pc11*/pc10*pc9*/pc8*/cbusack)
      then ACK4RW
      with
         statussel := ?;
         run := ?;
         bmacsel := ?;
         regsel := ?;
         memsel := ?;

      endwith
      else ACK3R
      with
         modesel := ?;
         attnenab := ?;
         statussel := ?;
         statusstb := ?;
         run := ?;
         bmacsel := ?;
         regsel := ?;
         memsel := ?;
         ack3 := 1;
         /iochrdy := 1;
      endwith;

   state ACK4RW :
      if /smrd then ACK4RW
      with
         statussel := ?;
         run := ?;
         bmacsel := ?;
         regsel := ?;
         memsel := ?;
      endwith
      else if /smwr then ACK4RW
      else IDLE
      with statussel := 1; endwith;

end state_diagram
```

Figure 4-34 PC Interface .OPL File
(Sheet 4 of 4)

```
#$ TOOL NSC
#$ TITLE  pc interface (state machine)
#$ TITLE
#$ DEVICE MAPL144
.i 21
.o 28
#$ PINS  29 aen:6 smwr:5 smrd:4 pc19:3 pc18:2 pc17:43 pc16:42
pc15:41 pc11:40 pc10:19 pc9:18 pc8:17 pc7:14 pc6:13 pc5:24
pcrst:25 cbusack:26 intb:27 statussel:20 modesel:21 attnen-
ab:22 run:23 bmacsel:7 ack3:10 memsel:11 regsel:12 statusstb:36
int:35 iochrdy:34
#$ NODES 3 pb2:47 pb1:46 pb0:45
.ilb aen smwr smrd pc19 pc18 pc17 pc16 pc15 pc11 pc10 pc9 pc8
pc7 pc6 pc5 pcrst cbusack intb pb2 pb1 pb0
.ob  pb2.d pb2.ce- pb1.d pb1.ce- pb0.d pb0.ce- statussel.j sta-
tussel.k- modesel.j modesel.k- attnenab.j attnenab.k- run.j
run.k- bmacsel.j bmacsel.k- ack3.j ack3.k- memsel.j memsel.k-
regsel.j regsel.k- statusstb.j statusstb.k- int.j int.k- io-
chrdy.j iochrdy.k-
.type f
.phase 111111111111111111111111111
.p 46
```

Figure 4-35 PC Interface Program Listing
(Sheet 1 of 2)

```
0-01101----------000 100000~~~~~~~~~~~~~~~~~~~~00
00-1101----------000 000010~~~~~~~~~~~~~~~~~~~~00
-----------------001 001000~~~~~~~~~~~~~~~~~~~~00
-------0001100----001 001000~~11~~~~~~~~~~~~~~~~
-------0001101----001 001000~~~~11~~~~~~~~~~~~~~
-------00011100---001 00100011~~~~~~~~~~~~~~~~~~
------------100---001 001000~~~~~~~~~~~~~~~~11~~~~
-------0001111----001 001000~~~~~~11~~~~~~~~~~~~~
---------------1--001 001000~~~~~~11~~~~~~~~~~~~~
-------00010------001 001000~~~~~~~~11~~~~~~~~~~~
-------0----------001 001000~~~~~~~~~~~~~~11~~~~~~
-------1----------001 001000~~~~~~~~~~~~11~~~~~~~~
-----------------010 00101001010101011101010l~~00
-------1----------011 101010~~~~~~~~~~~~~~~~~~~~~~
--------1---------011 101010~~~~~~~~~~~~~~~~~~~~~~
---------1--------011 101010~~~~~~~~~~~~~~~~~~~~~~
----------0-------011 101010~~~~~~~~~~~~~~~~~~~~~~
----------1------011 101010~~~~~~~~~~~~~~~~~~~~~~
----------------0-011 101010~~~~~~~~~~~~~~~~~~~~~~
-------00010----1-011 00101001010101011101010l~~00
-------0----------100 100010~~~~~~~~~~~~~~11~~~~~~
-------1----------100 100010~~~~~~~~~~~~11~~~~~~~~
-----------------100 100010~~~~~~~~~~~~~~~~~~~~00
-----------------101 10100001010101011101010l~~00
-------0001100----100 100010~~11~~~~~~~~~~~~~~~~
-------0001101----100 100010~~~~11~~~~~~~~~~~~~~
-------00011100---100 10001011~~~~~~~~~~~~~~~~~~
------------100---100 100010~~~~~~~~~~~~~~~~11~~~~
-------0001111----100 100010~~~~~~11~~~~~~~~~~~~~
---------------1--100 100010~~~~~~11~~~~~~~~~~~~~
-------00010------100 100010~~~~~~~~11~~~~~~~~~~~
--------1---------110 10101001~~~~0101~~0101~~~~~~
---------1--------110 10101001~~~~0101~~0101~~~~~~
----------0-------110 10101001~~~~0101~~0101~~~~~~
----------1------110 10101001~~~~0101~~0101~~~~~~
----------------0-110 10101001~~~~0101~~0101~~~~~~
-------00010----1-110 10100001010101011101010l~~00
--0---------------111 10101001~~~~0101~~0101~~~~~~
-01---------------111 101010~~~~~~~~~~~~~~~~~~~~~~
-11---------------111 00000011~~~~~~~~~~~~~~~~~~~~
.e
```

Figure 4-35 PC Interface Program Listing
(Sheet 2 of 2)

CLK	System Clock.
Outputs	
STATUSSEL	Output Enable for the STATUS Register.
STATUSSTB	Chip Select for the STATUS Register.
MODESEL	Chip Select for the MODE Register.
ATTNENAB	Chip Select for the FUNCTION Register.
BMACSEL	BMAC peripheral Chip Select.
RUN	Reset all state machines.
MEMSEL	Accessing the board under a memory select.
REGSEL	Accessing the board under a register select.
ACK3	Third state in the Access Control State Machine. Used for memory selects.
INT	Interrupt Control.
IOCHRDY	I/O Channel Ready.

CHAPTER 5

MAPL Support Tools

5.1 Introduction

There are a wide variety of third-party programmers and development software tools available to support National Semiconductor's PLDs, including the new MAPL family. With continual additions and changes, it is not possible to have the latest support list in this databook. The list in this section details all current or scheduled software and programmer support for the MAPL family.

5.2 Programmer Approvals

National Semiconductor has formal evaluation and certification procedures for PLD programmers. A programmer must fully meet our device programming specifications and pass extensive qualification testing in order to gain approval. To maintain device quality and avoid high programming fallout, only NSC approved programming equipment should be used.

5.3 PLCC Socket Adapters

As PLDs move into higher pin count PLCC packages, the choice of programming tools becomes more restricted. A few programmers are able to support high pin count PLCC packages, but many lower cost units have a 28-pin DIP capacity. It is still possible to program a MAPL device with such programmers by using a DIP to PLCC socket adapter. All approved DIP only programmers have been tested with such an adapter. Availability of these socket adapters is detailed below:

PROCON Technologies
(408) 246-4456 • FAX (408) 246-4435
PN #325-28-1228-DIP (Auto Eject)
1333 Lawrence Expressway, Suite 207
Santa Clara, CA 95051

Emulation Technologies
(408) 982-0660 • FAX (408) 982-0664
PN #AS-28-28-DIP-6 (Lidded)
2344 Walsh Avenue, Bldg. F
Santa Clara, CA 95051

5.4 MAPL Development Software

Package	Vendor	Version
OPAL	National Semiconductor	1.00
ABEL-4	Data I/O	4
CUPL	Logical Devices	Call vendor
Workview	ViewLogic	Call vendor

5.5 MAPL Programmer Support

Vendor	Programmer	Adapter	Revision
BP Microsystems	PLD 1128		
DATA I/O	Unisite	—	3.3
	2900	—	1.4
	29B	303A-011A	Scheduled
Elan	Series 5000	145	Scheduled
GP Industrial	AP100	P600	
	XP2M	M3	1.0
Logical Devices	Allpro-88	—	2.1
SMS	Sprint Plus	—	Scheduled
	Expert	—	Scheduled
Stag	ZL30A	—	42
	3000	ZM3000	12
System General	85A	AD19-01	2.0